

TCG PC Client Platform TPM Profile Specification for TPM 2.0

Version 1.06
Revision 32
February 8, 2024

Contact: admin@trustedcomputinggroup.org

Public Review

Work in Progress

This document is an intermediate draft for comment only and is subject to change without notice. Readers should not design products based on this document.

DISCLAIMERS, NOTICES, AND LICENSE TERMS

THIS SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Without limitation, TCG disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

This document is copyrighted by Trusted Computing Group (TCG), and no license, express or implied, is granted herein other than as follows: You may not copy or reproduce the document or distribute it to others without written permission from TCG, except that you may freely do so for the purposes of (a) examining or implementing TCG specifications or (b) developing, testing, or promoting information technology standards and best practices, so long as you distribute the document with these disclaimers, notices, and license terms.

Contact the Trusted Computing Group at www.trustedcomputinggroup.org for information on specification licensing through membership agreements.

Any marks and brands contained herein are the property of their respective owners.

CHANGE HISTORY

REVISION	DATE	DESCRIPTION
0.43	January 26, 2015	<ul style="list-style-type: none"> Initial Release of Version
1.03	May 22, 2017	<ul style="list-style-type: none"> Updated Command, Algorithm and Curve Tables based on Library 1.38 Incorporated prior errata Added GPIO functionality Added I2C interface
1.04	February 2020	<ul style="list-style-type: none"> Updated Algorithm and Curve Tables Added Enhanced Peripheral Interface functionality
1.05		<ul style="list-style-type: none"> Added mandatory support for RSA 3072-bit keys Made TPM_ALG_ECDSA and TPM_ALG_ECSCHEMORR optional Made TPM_ECC_BN_P256 optional Made TPM2_Commit optional Added new commands from TPM2 Library Specification as optional
1.06		<ul style="list-style-type: none">

DRAFT

CONTENTS

- DISCLAIMERS, NOTICES, AND LICENSE TERMS 1
- CHANGE HISTORY 2
- 1 SCOPE 9
 - 1.1 Key Words 9
 - 1.2 Statement Type 9
- 2 TPM Requirements General Introduction 10
 - 2.1 Terminology 10
 - 2.2 Division of Documentation 11
- 3 Summary of TPM Features to Support the PC Client 12
 - 3.1 Register Definitions 12
 - 3.2 Locality 12
 - 3.3 Interface Type 12
 - 3.4 Locality Resettable PCRs 13
 - 3.5 Minimum Amount of NV Storage 13
 - 3.6 Minimum Number of PCRs 13
- 4 TPM Attributes 14
 - 4.1 PC Client TPM Minimums and Maximums 14
 - 4.2 PC Client Algorithms 15
 - 4.3 PC Client Curves 17
 - 4.4 Physical Presence 17
 - 4.5 Non-volatile Storage for NV Indexes and Persistent Objects 17
 - 4.5.1 NV Storage Requirements for Persistent Objects 17
 - 4.5.2 NV Storage Requirements for NV Indexes 18
 - 4.5.3 Endorsement Key Certificates 19
 - 4.5.4 TPM NV Indices for Counters and PINs 20
 - 4.5.5 General Purpose I/O (GPIO) 21
 - 4.6 PCR Requirements 26
 - 4.6.1 PCR Attributes 27
 - 4.6.2 PCR Initial and Reset Values 28
 - 4.7 Power Management 29
 - 4.8 Self-Test Requirements 30
 - 4.9 Firmware Upgrade 30
- 5 TPM Capabilities and Commands 32
 - 5.1 Platform Specific Requirements 32
 - 5.1.1 TPM Handles, Objects and Contexts 32

5.1.2	Authenticated Countdown Timer (ACT).....	32
5.2	Command Table	32
5.3	Locality-Controlled Functions	38
5.3.1	D-RTM Execution Sequence	38
5.3.2	S-HCRTM Sequence Before TPM2_Startup and TPM2_Startup without S-HCRTM.....	42
5.3.3	Timing and Protocol	42
6	TPM Software Interface	44
6.1	Interface Type	44
6.2	Locality.....	44
6.2.1	TPM Locality Levels	44
6.2.2	Locality Uses.....	46
6.3	TPM Register Space	47
6.3.1	TPM Register Space Decode	47
6.3.2	Register Space Addresses	49
6.4	System Interaction and Flows	55
6.4.1	FIFO Configuration Registers.....	55
6.4.2	Interface Identifier Register.....	55
6.5	TPM's Software Interaction	63
6.5.1	Interface-Agnostic functions.....	64
6.5.2	FIFO Interface Requirements	73
6.5.3	CRB Interface Requirements.....	103
6.6	Interrupts.....	122
6.6.1	FIFO Interrupts.....	123
6.6.2	CRB Interrupts	126
7	TPM Hardware.....	129
7.1	FIFO Interface Locality Usage per Register.....	129
7.2	CRB Interface Locality Usage Per Register.....	130
7.3	TPM LPC Hardware Protocol	132
7.3.1	LPC Locality Cycles for TPM Interface.....	132
7.4	SPI Hardware Protocol.....	133
7.4.1	Clocking.....	134
7.4.2	Electrical Specification	134
7.4.3	SPI Interrupts	137
7.4.4	Legacy I/O.....	137
7.4.5	Flow Control	137
7.4.6	SPI Bit Protocol.....	141

7.5	TPM Byte Ordering.....	142
7.6	Reset Timing.....	143
8	I2C Interface Definition.....	145
8.1	TPM I2C Interface Requirements.....	145
8.1.1	Bus speed.....	145
8.1.2	I2C Device address.....	145
8.1.3	Fast turnaround.....	145
8.1.4	Data rate synchronization.....	146
8.1.5	Supply voltage.....	146
8.1.6	Pull-up resistors.....	146
8.1.7	Host interrupt.....	146
8.1.8	Availability after reset.....	146
8.1.9	Locality support.....	147
8.1.10	GUARD_TIME.....	147
8.2	Communication Protocol Fundamentals.....	147
8.2.1	Layer Model.....	148
8.2.2	Physical Layer I2C.....	148
8.3	Physical Layer TCG-I2C.....	155
8.3.1	Byte Ordering.....	156
8.3.2	Overruns.....	156
8.3.3	Handling of Multi-Byte Registers.....	156
8.3.4	I2C-TPM Localities.....	157
8.3.5	I2C-TPM Registers.....	157
8.3.6	Interface Locality Usage per Register.....	167
8.3.7	TCG-I2C Protocol Usage Scenarios.....	168
9	TPM Hardware Implementation.....	172
9.1	TPM Packaging.....	172
10	Platform Specific Hardware Implementation of a TPM in a PC Client Platform.....	179
10.1	Electrical Connections for a Discrete TPM.....	179
10.1.1	SPI Platform Design Notes.....	179
10.1.2	Software Interface to SPI-TPM.....	179
10.1.3	Software Command Interface to SPI-TPM.....	180
10.2	NV Index Provisioning for Platform Firmware Supported Features.....	180
11	References.....	182

FIGURES

Figure 1 — Overview of D-RTM Measurement Sequence	40
Figure 2 — PC Client Initialization Sequence	68
Figure 3 — State Transition Diagram	87
Figure 4 — TPM State Diagram for CRB Interface	117
Figure 5 — Timing Diagram	136
Figure 6 — Clock Timing Diagram	137
Figure 7 — Example Read transaction with a WAIT state	140
Figure 8 — Example of WRITE transaction with Wait state	140
Figure 9 — Layer Model	148
Figure 10 — Register write sequence on the I2C layer	151
Figure 11 — Example Address NACK in a register write sequence on the I2C layer	152
Figure 12 — Register read sequence on the I2C layer	153
Figure 13 — Register read sequence on the I2C layer using repeated START (Sr)	154
Figure 14 — Register read sequence with GUARD_TIME write after read on the I2C layer	155
Figure 15 — Write / Read TPM_ACCESS register without locality selection	168
Figure 16 — Write / Read TPM_ACCESS register from Locality 0	168
Figure 17 — Write / Read TPM_ACCESS register from Locality 0 and 2	169
Figure 18 — Read / Write TPM_STS register(s) from Locality 0	170
Figure 19 — Read TPM_DATA_FIFO	170
Figure 20 — Write TPM_DATA_FIFO	171
Figure 21 — TPM Combo TSSOP-28 Pin Out	173
Figure 22 — TPM SPI QFN-32 Pin out	173

TABLES

Table 1 — PTP Revision to TPM Library Revision	10
Table 2 — TPM Requirements.....	14
Table 3 — PC Client TPM Algorithms.....	16
Table 4 — TPM Mandatory and Optional Curves	17
Table 5 — Requested NV Index List.....	19
Table 6 — EK Certificate Allowed Combinations.....	20
Table 7 — EPI Configuration Index Definition	23
Table 8 — TPMA_NV Support Requirements for EPI Memory Devices	25
Table 9 — TPM_NT Support for EPI Memory Devices.....	26
Table 10 — PCR Attributes	28
Table 11 — PCR Initial and Reset Values	28
Table 12 — Mandatory/Optional TPM Commands	32
Table 13 — Locality Address Definitions	45
Table 14 — Relationship between Locality and Locality Attribute	46
Table 15 — Example Bit-to-Address Mapping.....	49
Table 16 — Allocation of Register Space for FIFO and CRB Access	50
Table 17 — DID/VID Register	55
Table 18 — RID Register	55
Table 19 — FIFO Interface Identifier Register	56
Table 20 — CRB Interface Identifier Register.....	59
Table 21 — CRB Historical Interface Versions	61
Table 22 —Command Timing	66
Table 23 — Definition of Interface Timeouts.....	67
Table 24 — TPM_DATA_CSUM_ENABLE Definition	71
Table 25 — TPM_DATA_CSUM Definition.....	71
Table 26 — Allocation of Register Space for FIFO TPM Access	73
Table 27 — Access Register.....	81
Table 28 — Status Register	88
Table 29 — Data FIFO Register	96
Table 30 — Interface Capability.....	96
Table 31 — State Transition Table	100
Table 32 — Address Allocation for CRB TPM Access	103
Table 33 — TPM_LOC_STATE Definition.....	108
Table 34 — TPM_LOC_CTRL_x Register Definition.....	109
Table 35 — TPM_LOC_CTRL_4 Register Definition.....	110
Table 36 — TPM_LOC_STS.....	111
Table 37 — TPM CRB Control Area Request.....	112
Table 38 — TPM CRB Control Area Status.....	114
Table 39 — TPM CRB Control Cancel	115
Table 40 — TPM CRB Control Start	116
Table 41 — CRB Interface State Transitions.....	120
Table 42 — FIFO Interrupt Enable.....	124
Table 43 — Interrupt Status.....	125
Table 44 — Interrupt Vector.....	126
Table 45 — CRB Interrupt Control	127
Table 46 — Interrupt Status	128
Table 47 — Register Behavior Based on Locality Setting for FIFO.....	129
Table 48 — Register Behavior Based on Locality Setting for CRB	131
Table 49 — LPC Locality Cycle TPM-Write for Accessing a TPM.....	133
Table 50 — LPC Cycle TPM-Read for Accessing a TPM.....	133
Table 51 — DC Specifications for 1.8V Supply Voltage	135
Table 52 — DC Specifications for 3.3V Supply Voltage	135
Table 53 — AC Electrical Specifications.....	136
Table 54 — SPI Bit Protocol	142
Table 55 — Register Behavior Based on Locality Setting for I2C	149

Table 56 — TPM Locality Selection Register 157

Table 57 — I2C-TPM Register Overview..... 157

Table 58 — TPM Locality Selection Register 160

Table 59 — Interrupt Enable 160

Table 60 — Interrupt Capability 161

Table 61 — Status Register 161

Table 62 — I2C Interface Capability Register..... 163

Table 63 — Data Checksum Enable Register 165

Table 64 — Data Checksum Register 166

Table 65 — I2C Device Address Register 167

Table 66 — TSSOP-28 Pin Assignments 174

Table 67 — TSSOP 28 I2C Pin Assignments..... 176

Table 68 — QFN-32 Pin Assignments..... 177

Table 69 — PFP Defined NV Extend Indexes 180

DRAFT

1 SCOPE

A TPM claiming adherence to this specification SHALL be compliant with a TPM *Library Specification; Family 2.0; Level 00; Revision 01.59* or later. A TPM compliant with this specification may be implemented in hardware, firmware or software. Hardware requirements may not apply to firmware or software TPMs.

1.1 Key Words

The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” in this document normative statements are to be interpreted as described in RFC-2119, “Key words for use in RFCs” to Indicate Requirement Levels.

1.2 Statement Type

Please note a very important distinction between different sections of text throughout this document. There are two distinctive kinds of text: informative comment and normative statements. Because most of the text in this specification will be of the kind normative statements, the authors have informally defined it as the default and, as such, have specifically called out text of the kind informative comment. They have done this by flagging the beginning and end of each informative comment and highlighting its text in gray. This means that unless text is specifically marked as of the kind informative comment, it can be considered a kind of normative statements.

2 TPM Requirements General Introduction

Start of informative comment

A TPM Library Specification define a TPM for use on any generic platform. Platform-specific functionality is defined in platform specifications such as this document.

End of informative comment

1. This document details the additional features that SHALL be implemented by a TPM for a PC Client platform.

Unless otherwise indicated, the features in this specification are based on a TPM Library Specification Family 2.0; Level 00; Revision 01.59 parts 1 through 3. The term TPM Library Specification is used to refer to these documents and the features they specify.

Table 1 — PTP Revision to TPM Library Revision describes the historical relationships between PTP versions and TPM Library Specification versions.

Table 1 — PTP Revision to TPM Library Revision

PTP Version	TPM Library Specification Version
PTP revision 0.43	Library Spec Revision 0.99
PTP version 1.03 revision 17	Library Spec Revision 1.38
PTP version 1.04 revision 37	Library Spec Revision 1.38
PTP version 1.05 Revision 14	Library Spec Revision 1.59

2.1 Terminology

Start of informative comment

The following terms are used as defined below throughout the document. All other terms are defined in the PC Client Platform Firmware Profile for TPM 2.0 Systems [3] or the TCG Glossary [10].

HASH_START: A successful write to a TPM's TPM_HASH_START interface register (FIFO) or a TPM_LOC_CTRL_4.TPM_HASH_START field (CRB) using the _TPM_Hash_Start signal as defined in a TPM 2.0 Library Specification.

HASH_DATA: A successful write to a TPM's TPM_HASH_DATA interface register (FIFO) or a TPM_LOC_CTRL_4.TPM_HASH_DATA field (CRB) using the _TPM_Hash_Data signal as defined in a TPM 2.0 Library Specification.

HASH_END: A successful write to a TPM's TPM_HASH_END interface register (FIFO) or a TPM_LOC_CTRL_4.TPM_HASH_END field (CRB) using the _TPM_Hash_End signal as defined in a TPM 2.0 Library Specification.

TPM Device Reset: the assertion of the _TPM_INIT hardware signal.

Platform Software: the source of the command, which may be an operating system driver or an application.

Platform Hardware: platform components including chipsets and associated microcode, and microprocessors and associated microcode.

Operating System, or OS: generic term for an operating system and its collection of drivers and services.

Static OS: the operating system that is loaded during the initial boot sequence of the platform from its platform reset.

Dynamic OS: an operating system that is loaded by the Static OS. There may be more than one Dynamic OS per Host Platform but only one can be loaded at a time. The Dynamic OS can be unloaded keeping the Static OS resident and operational.

Read: a transaction where the calling entity requests and receives data from a specified register or buffer in a TPM.

Write: a transaction where the calling entity sends data to a register or buffer in a TPM.

PC Client Platform Implementation Specification: the combination of the PC Client Platform Firmware Profile Specification for TPM 2.0 Systems [3], the TCG ACPI Specification [11], and the PC Client Physical Presence Interface Specification [12].

The following conventions are used to represent values of fields in this document:

Any field which contains a value is represented in hexadecimal format (e.g., B0h).

Any field which contains a bitfield is represented in binary format (e.g., 0001b).

End of informative comment

2.2 Division of Documentation

Start of informative comment

The PC Client Specifications are divided into two documents:

This specification, the *PC Client Specific Platform TPM Profile for TPM 2.0*, discusses the specifics regarding the requirements of a TPM for PC Client but only the requirements for a TPM itself, not the requirements for a platform incorporating a TPM. This document discusses the details of what interfaces and protocols are used to communicate with a TPM and the platform-specific set of requirements. This document includes the definitions of the items identified in a TPM Library specification as “Platform Specific” such as the minimum number of PCRs required and NV Storage available. The target audience for this document is a TPM manufacturers but platform manufacturers should review it as well to understand security constraints for a platform incorporating a TPM.

The *PC Client Platform Implementation Specification* specifies the requirements for a TPM as it is implemented on the platform. Issues such as TPM, platform and bios provisioning, usage of TPM to record measurements of platform code, PCR mapping, functional interfaces, and interfaces are discussed. The target audience for this document is platform manufacturers.

End of informative comment

3 Summary of TPM Features to Support the PC Client

3.1 Register Definitions

Start of informative comment

This specification identifies the various registers that allow communication between a TPM and platform hardware and software.

End of informative comment

3.2 Locality

Start of informative comment

“Locality” is an assertion to a TPM that a command's source is associated with a particular component. Locality can be thought of as a hardware-based authorization. A TPM is not actually aware of the nature of the relationship between the locality and the component. The ability to reset and extend notwithstanding, it is important to note that, from a PCR “usage” perspective, there is no hierarchical relationship between different localities. A TPM simply enforces locality restrictions on TPM assets (such as PCR or SEALED blobs). For example, PCR attribute settings may allow a component associated with Locality 4 to reset PCR associated with Locality 2; and a SEALED blob may use an authorization policy to allow it to be accessed from locality 2 but not from locality 4.

The protection and separation of the localities (and therefore the association with the associated components) is entirely the responsibility of the platform components. Platform components, including the OS, may provide the separation of localities using protection mechanisms such as virtual memory or paging.

For the FIFO and CRB interfaces, assertion of locality is done by interacting with a TPM at specified blocks of address ranges. Each locality is assigned an address range, and, when a command is received at the address range associated with a locality, a TPM sets a TPM's internal *localityModifier* value to the indicated locality value.

Note on convention for using the term locality: When referring to localities in general the term locality will be lower case (i.e., starts with an ‘l’.) When discussing a specific locality, the term locality will be capitalized (i.e., Locality 0 does something.) When using a phrase such as: “executes at Locality 0”, this means the command is sent to the memory-mapped TPM addresses defined for Locality 0, and the platform components that enforce access to a TPM have authorized that command be sent from that component to that address.

The PC Client TPM interface defines the attributes and use of five localities (Localities 0 – 4). The nominal association of these localities is:

Locality 4: Usually associated with the CPU executing microcode. This is used to establish the Dynamic RTM.

Note: See the *PC Client Platform Firmware Profile for TPM 2.0 Systems* [3] for the definition of Dynamic RTM.

Locality 3: Auxiliary components. Use of this is optional and, if used, it is implementation dependent.

Locality 2: Dynamically Launched OS (Dynamic OS) “runtime” environment.

Locality 1: An environment for use by the Dynamic OS.

Locality 0: The Static RTM, its chain of trust and its environment.

Note: These associations are arbitrary and depend on the system implementation.

End of informative comment

3.3 Interface Type

Start of informative comment

This specification defines a new Software interface to a TPM for TPM 2.0, in addition to the FIFO interface. This interface, the Command Response Buffer Interface, has been defined so that it may be implemented in a TPM which also contains a FIFO interface. The CRB Interface is intended to be physical-bus agnostic, so that it could be implemented on an LPC or SPI interface, as specified in this specification or on another physical interface not

specified. For a TPM to be compliant with this specification, however, it is required to implement at least one of the interfaces defined by this specification.

The physical register spaces for both FIFO and CRB are specified in Section 6.3 TPM Register Space. Register space with functions common to both interfaces is specified in Section 6.5.1 Interface-Agnostic functions. The behavior of the CRB Interface is specified in Section 6.5.3 CRB Interface Requirements. In the subsequent sections, functionality which is interface-independent precedes the interface-specific functionality. Where a function is common to both interfaces, but there are interface-specific requirements, the requirements are documented in the interface-specific section. For example, the concepts of locality are common to both interfaces, but the mechanisms to invoke locality are interface specific.

End of informative comment

3.4 Locality Resettable PCRs

Start of informative comment

Resettable PCR, except for PCR 16 and PCR 23, are a set of PCRs for use by the Dynamic RTM and its chain of trust. Access to these PCR is controlled by the various locality indicators.

End of informative comment

3.5 Minimum Amount of NV Storage

Start of informative comment

A *TPM 2.0 Library Specification* [7] provides for a general-purpose area of non-volatile storage for use by the platforms as well as for storage of persistent objects. This is different from TPM 1.2, in that the non-volatile storage for persistent objects was TPM vendor implementation specific. The definition of this area is the purview of the various platform specific specifications. This specification defines the minimum amount required for the PC Client platform.

End of informative comment

3.6 Minimum Number of PCRs

Start of informative comment

A *TPM 2.0 Library Specification* [7] allows the platform specific specifications to require a minimum number of PCRs and to allocate usage for them based on the needs and the environment of the platform. Additionally, a *TPM 2.0 Library Specification* allows the platform specific specification to define whether authorization is required to extend or reset PCRs. As PC Client platforms have stringent boot time requirements, this specification does not use authorization for operations on PCRs that will be used in the platform boot process (TPM2_PCR_Extend).

End of informative comment

4 TPM Attributes

4.1 PC Client TPM Minimums and Maximums

Start of informative comment

A TPM *Library Specification* allows a variety of implementations to be defined from the superset of functionality contained within the library. This section defines the minimum and maximum requirements for a PC Client TPM of those attributes in the Library Specification that are left to the Platform specification to define.

Table 1 contains the names of the property types, as defined in a TPM Library Specification Part 2, Section TPM_PT, which must be specified by a platform profile. Table 1 defines the value returned by a TPM2_Get_Capability query to each PT name for a PC Client TPM. The values stated in Table 2 TPM Requirements may be greater than the minimum or less than the maximum number indicated. If a number does not have a “min” or “max” designation, it is defined to be exactly that number. The values stated in Table 2 TPM Requirements are the capabilities of a TPM prior to provisioning.

Mandatory support for RSA 3072-bit keys, added in PTP version 1.05, impacts TPM performance in several ways, including increased time to generate keys, less space for persistent objects and more frequent loading and flushing of keys to and from transient memory. This specification mandates support for only two persistently loaded RSA 3072-bit keys. If software attempts to persist additional RSA 3072-bit keys, a TPM may respond with an error. If, however, software loads an RSA 2048-bit key and attempts to persist it, this operation should succeed without an error.

End of informative comment

1. A TPM claiming conformance to this specification SHALL support the minimum and maximum requirements defined in Table 2 — TPM Requirements.
2. A TPM SHALL report all defined TPM_PT properties defined in a TPM Library Specification in response to a TPM2_GetCapability command.

Table 2 — TPM Requirements

Capability Name	Returned Value	Description
TPM_PT_HR_TRANSIENT_MIN	3 min	The minimum number of transient objects that can be held in a TPM RAM. A TPM SHALL support at least 2 RSA 3072-bit keys.
TPM_PT_HR_PERSISTENT_MIN	7 min	The minimum number of persistent objects that can be held in TPM NV Memory. When calculating this number, the following example allocations were used: <ul style="list-style-type: none"> • 3 slots intended for root keys (PPK, SRK, 1 EKs), • 3 slots intended for OS/application usage, • 1 slot intended for the platform hierarchy. A TPM SHALL support at least 2 persistent RSA 3072-bit keys.
TPM_PT_HR_LOADED_MIN	3 min	The minimum number of authorization sessions that can be held in TPM RAM
TPM_PT_ACTIVE_SESSIONS_MAX	64 min	The minimum number of authorization sessions that may be active concurrently.
TPM_PT_PCR_COUNT	24 min	the number of PCR implemented in a bank
TPM_PT_PCR_SELECT_MIN	3 min	The minimum number of octets in a TPMS_PCR_SELECT.sizeOfSelect
TPM_PT_NV_COUNTERS_MAX	See Section 4.5.4	The maximum number of NV Indexes that can have a TPM_NT_COUNTER attribute SET. Note: See also TPM Library Specification Part 2 TPM_PT_NV_COUNTERS_MAX section.

TPM_PT_NV_INDEX_MAX	1600 min	The maximum decimal size of an NV Index data area Note: This is the size of an X.509 certificate signed with an RSA key and its authorization. The size specified here is the smallest maximum size a TPM vendor must support. TPM vendors may support larger sizes. The specific number and type of NV indices mandated by this specification is defined in Section 4.5.1.
TPM_PT_PS_FAMILY_INDICATOR	0x00000001	PC Client Platform TPM Specification Family
TPM_PT_PS_LEVEL	0x00000000	PC Client Platform TPM Specification Level 00
TPM_PT_PS_REVISION	0x00000106	TPM_PT_PS_REVISION SHALL be formatted as follows: 0xAABBCCDD where: <ul style="list-style-type: none"> • AA and BB SHALL be 00's, • CC SHALL be the major revision of the specification, • DD SHALL be the minor revision of the specification. e.g., 0x00000104 for a PTP Specification revision of 1.04
TPM_PT_PS_DAY_OF_YEAR	0x00000000	The platform-specific specification day of year.
TPM_PT_PS_YEAR	0x00000000	The platform-specific specification year.
TPM_PT_VENDOR_STRING_1	Defined by vendor	This field may be defined as an ASCII string no more than 4 characters. Unused characters SHALL be NULL
TPM_PT_VENDOR_STRING_2	Defined by vendor	This field may be defined as an ASCII string no more than 4 characters. Unused characters SHALL be NULL. This field is concatenated to TPM_PT_VENDOR_STRING_1
TPM_PT_VENDOR_STRING_3	Defined by vendor	This field may be defined as an ASCII string no more than 4 characters. Unused characters SHALL be NULL. This field is concatenated to TPM_PT_VENDOR_STRING_2
TPM_PT_VENDOR_STRING_4	Defined by vendor	This field may be defined as an ASCII string no more than 4 characters. Unused characters SHALL be NULL. This field is concatenated to TPM_PT_VENDOR_STRING_3
TPM_PT_VENDOR_TPM_TYPE	0x00000000	Reserved, not used.
TPM_PT_FIRMWARE_VERSION_1	Defined by vendor	The upper 16 bits of this field SHALL contain a TPM major firmware version (Version Major) The lower 16 bits of this field SHALL contain a TPM minor firmware version (Version Minor)
TPM_PT_FIRMWARE_VERSION_2	Defined by vendor	This field MAY be used as an extension to TPM_PT_FIRMWARE_VERSION_1. If not used, this field SHALL be 0.
NUM_POLICY_PCR_GROUP	0	number of PCR groups that have individual policies
NUM_AUTHVALUE_PCR_GROUP	0	number of PCR groups that have individual authorization values
TPM_PT_NV_BUFFER_MAX	512 min	The smallest permissible buffer size for the data in a TPM2_NV_Read or TPM2_NV_Write command
NV_MEMORY_SIZE	Defined by vendor	See Section 4.5.1 for further details

4.2 PC Client Algorithms

Start of informative comment

All algorithm identifiers listed in Table 2 are mandatory for a PC Client TPM. Some algorithms listed below are not explicitly selectable as they are supporting algorithms needed for a higher-level function, e.g.,

TPM_ALG_ECSCNORR is required for TPM_ALG_ECDSA. Algorithms not explicitly listed are optional and may be required if an optional command is implemented by a TPM. A hash algorithm for the purposes of this specification is defined to be an algorithm with a Type limited to “H” in the TCG Algorithm Registry.

End of informative comment

To be compliant to this specification:

1. A TPM SHALL support algorithms listed in Table 3 as Mandatory (M),
2. A TPM SHOULD support algorithms listed in Table 3 as Recommended (R).
3. A TPM MAY support algorithms listed in Table 3 as Optional (O).
4. Algorithms listed as Deprecated (D) MAY be removed in future.
5. Algorithms listed as Not Allowed (N) SHALL NOT be supported.

Table 3 — PC Client TPM Algorithms

Algorithm ID	M/R/O /D	Comments
TPM_ALG_RSA	M	Support for 2048-bit and 3072-bit keys is required; a TPM SHOULD NOT support 1024-bit keys. RSA 3072 is mandatory as of PTP version 1.05. A TPM SHALL support the default public exponent defined in a TPM Library Specification Part 2 TPMS_RSA_PARMS.
TPM_ALG_TDES	N	This algorithm was made Not Allowed in PTP 1.06
TPM_ALG_SHA1	O	This is a hash algorithm. Made optional as of PTP 1.06 Mandatory algorithms for PCRs are defined in Section 4.6
TPM_ALG_HMAC	M	
TPM_ALG_AES	M	SHALL support 128- and 256-bit keys and TPM_ALG_CFB at a minimum. TPM_ALG_ECB SHOULD NOT be used. AES 256 support is mandatory as of PTP 1.03
TPM_ALG_MGF1	M	
TPM_ALG_KEYEDHASH	M	
TPM_ALG_XOR	M	
TPM_ALG_SHA256	M	This is a hash algorithm. Mandatory algorithms for PCRs are defined in Section 4.6
TPM_ALG_SHA384	M	This is a hash algorithm, mandatory as of PTP version 1.04. Mandatory algorithms for PCRs are defined in Section 4.6
TPM_ALG_SHA512	O	This is a hash algorithm
TPM_ALG_NULL	M	
TPM_ALG_SM3_256	O	This is a hash algorithm
TPM_ALG_SM4	O	
TPM_ALG_RSASSA	M	
TPM_ALG_RSAES	M	
TPM_ALG_RSAPSS	M	
TPM_ALG_OAEP	M	
TPM_ALG_ECDSA	M	
TPM_ALG_ECDH	M	
TPM_ALG_ECDA	O	Note – change made to the implementation in TPM Library Specification Family 2.0 revision 1.35. This algorithm was made optional in PTP 1.05.
TPM_ALG_SM2	O	
TPM_ALG_ECSCNORR	O	Note – change made to the implementation in TPM Library Specification Family 2.0 revision 1.35. This algorithm was made optional in PTP 1.05.
TPM_ALG_ECC	M	See Section 4.3 for required curves

TPM_ALG_ECMQV	O	
TPM_ALG_SYMCIPHER	M	
TPM_ALG_CAMELLIA	O	
TPM_ALG_SHA3_256	O	This is a hash algorithm
TPM_ALG_SHA3_384	O	This is a hash algorithm
TPM_ALG_SHA3_512	O	This is a hash algorithm

4.3 PC Client Curves

1. To be compliant to this specification:
 - a. A TPM SHALL implement the curves listed in Table 4 as Mandatory (M).
 - b. A TPM SHOULD implement the curves listed in Table 4 as Recommended (R).
 - c. A TPM MAY implement the curves listed in
 - d. Table 4 as Optional (O).

Table 4 — TPM Mandatory and Optional Curves

Curve Identifier	M/R/O/	Comments
TPM_ECC_NIST_P256	M	
TPM_ECC_BN_P256	O	SHOULD only be used to support ECDAA. Made optional as of PTP 1.05
TPM_ECC_NIST_P384	M	Mandatory as of Platform TPM Profile 1.04
TPM_ECC_SM2_P256	O	Included in this table (Table 4) as of PTP 1.06

4.4 Physical Presence

Start of informative comment

Physical Presence is not required for a PC Client TPM to be compliant to this specification.

End of informative comment

4.5 Non-volatile Storage for NV Indexes and Persistent Objects

Start of informative comment

The non-volatile (NV) Storage provides a small general-purpose data storage area for persistent data. A TPM provides the ability to add access control to this area for security or privacy. This area is organized and addressed using indices.

While this area provides a general-purpose storage area for interoperability, it also provides a storage location for persistent objects used by a TPM. To accommodate storage of persistent objects and Certificates for some of these objects, some index values are reserved. These values are defined in the *Registry of Reserved TPM 2.0 Handles and Localities* [13]. A reserved index value is an index which has been defined by TCG, but for which there is no requirement to implement the value, e.g., the Endorsement Key Credential index. A reserved index value, if not implemented must not be used for a different purpose than defined.

A TPM will enforce any defined attributes for the NV storage.

TPM 2.0 allows for NV Indices to be defined by the platform OEM or the Owner. Platform indices may only be created using Platform Authorization. Owner indices are created using Owner Authorization. Platform NV is distinguished from Owner NV by an attribute in the NV Public area, TPMA_NV_PLATFORMCREATE. If this attribute is set, the NV Index was created using Platform Authorization.

End of informative comment

4.5.1 NV Storage Requirements for Persistent Objects

Start of informative comment

A TPM Library Specification allows implementations to pool the NV memory used for persistent objects with the memory used for NV Indexes. A TPM Library reference implementation allows definition of NV Indexes until a TPM runs out of NV space but reserves enough NV memory for the minimum number of persistent objects to be created. The NV Indexes and persistent objects may come from the same NV memory pool, or they may come from different memory pools. A caller may discover a TPM's implementation by reading TPM_PT_MEMORY. If TPM_PT_MEMORY is SET, the memory is pooled between persistent objects and NV Indexes. The options vendors can choose include using a common pool with some reserved area, or common pool with no reserved area or completely separate pools.

If a TPM implements a shared pool of memory without reserved space for persistent objects, a TPM will return TPM_RC_NV_SPACE if the NV memory pool has been consumed by NV Indexes.

If TPMA_MEMORY.sharedNV is CLEAR, a TPM has separate pools for NV Indexes and Persistent Objects. If TPMA_MEMORY.sharedNV is SET, a TPM may have a common pool or reserve a minimum amount of NV for persistent objects.

End of informative comment

1. A read to TPM_PT_MEMORY SHALL return TPMA_MEMORY:
 - a. sharedNV MAY be SET or CLEAR,
 - b. sharedRAM SHALL be CLEAR, and
 - c. objectCopiedToRam MAY be SET or CLEAR.

4.5.2 NV Storage Requirements for NV Indexes

Start of informative comment

A TPM contains NV Storage that a TPM uses for internal purposes and some that is made available to callers to a TPM in the form of NV Indexes and persistent objects. A TPM specification does not define the required lifetime, or endurance, of the NV memory. There are several factors that affect the endurance, including operating temperature, operating voltage, and the PC Client platform within which a TPM is installed. This specification defines endurance of a TPM NV in terms of the minimum amount of data written to a TPM over its lifetime that a TPM is required to support.

Providing an adequate minimum amount of storage space is difficult to predict based on future and unspecified use of the platform. However, it is prudent to provide for some minimum and predictable amount of storage to allow processes to budget their allocation. For this reason, this specification defines the minimum amount of storage and number of indices that a TPM must implement. A TPM when it is delivered to a platform manufacturer is required to support a minimum amount of NV which can be configured to support any type of index defined in a TPM 2.0 Library Specification.

This specification does not define how a TPM vendor must organize a TPM's NV Storage. A TPM vendor may organize a TPM's NV Storage in such a way that the total amount of storage, minus the overhead required to implement individual indices, is allocated dynamically.

However a TPM is implemented, it is expected to provide flexibility in allocation of indices and storage allocation to the indices. A TPM is expected to provide a malloc()-style allocation of the NV storage area rather than provide a fixed size for each index. For example, a caller could define 9 indices of 1 byte each and a single index that consumes the remaining available space. Alternatively, a caller could define 10 indices of equal size. A TPM with a flexible implementation would allow either extreme.

The minimum NV storage is defined based on an example allocation and only includes the data area of NV indices. A TPM might treat all non-volatile entities as part of the same pool of NV memory. Depending on implementation, TPMs may include persistently held keys and certificates in the same pool of NV memory defined for NV indices. The minimum amount of NV Storage is calculated based on indices defined using TPM2_NV_DefineSpace which consume an NV index, exclusive of the overhead. The mandatory minimum size does not include storage of any other persistent object, data or code that requires or consumes non-volatile memory. If a TPM includes persistent storage of objects and Endorsement Certificates in the same pool of NV memory as NV Indices, a TPM vendor will need to account for the size of the objects and Endorsement Certificates, and the NV index overhead in calculating the required size of their NV memory pool. Table 5 below includes a sample set of indices identified by TPM vendors, Platform OEMs,

and OS vendors, but is not exhaustive. Table 5 does not include NV indices for Endorsement Certificates pre-provisioned by a TPM vendor.

Note: NV Indices for EK Certificates are not included in the minimum size calculation. TPM vendors must account for that space separately.

End of informative comment

1. A TPM SHALL be capable of supporting a minimum of 100KB written to TPM non-volatile memory over the life of a TPM.
2. A TPM SHALL support allocation of at least 68 indices, with a total minimum data size of 3834 Bytes (decimal). Table 5 provides a non-normative example of how this space could be partitioned.

Note: The sizes stated in the table are the maximum size of the data area only and do not address the overhead. TPM vendors need to account for the overhead.

Table 5 — Requested NV Index List

Requestor/Spec	Usage	Number of Indices	Data Size per Index (Bytes)
Platform Owner	Application Certificate	1	1280
TCG DRTM	General Purpose	1	104
TCG DRTM	General Purpose	1	70
Chipset Vendor	General Purpose	1	8
TCG PTP	NV Counters	6	8
TCG PTP	NV Pin	2	8
Software	NV Extend	1	48
User	General Purpose Index	1	48
Platform	Key storage	2	256
User	Object Authorization	2	50
OS	Secure storage	50	32

4.5.3 Endorsement Key Certificates

Start of informative comment

A TPM vendor may preprovision Endorsement Key certificates in TPM NV. This specification does not require preprovisioned certificates. If a TPM vendor chooses to provision certificates, the vendor may do so using TCG defined API's or vendor proprietary API's. Reserved handles for EK Certificates and other NV Indices related to the EK, such as the EK template, are defined in the TCG Registry of TPM 2.0 Handles and Localities [13]. The format of the certificate is defined in the TCG Endorsement Key Credential Profile for TPM Family 2.0 [14]; level 0, Specification Version 12 or later. The requirements in the following sub-section apply only if a certificate is pre-provisioned by the vendor.

End of informative comment

4.5.3.1 Preprovisioned EK Certificates

Start of informative comment

TPM vendors may utilize a TPM2_NV_DefineSpace and TPM2_NV_Write commands to create the index used to store the EK Certificate and to write it. TPM vendors may also utilize manufacturing processes to pre-provision EK

Certificates. There may be cases where a vendor proprietary provisioning process results in a certificate which cannot be deleted. In this case, the attributes TPMA_NV_POLICY_DELETE and TPMA_NV_POLICYWRITE may be set and TPMA_NV_PPWRITE may be cleared. This allows for a standard TCG error in the event someone attempts to clear the EK certificate index. TPM vendors should take care to ensure that the design of their certificate authority considers the strength and algorithm family for EKs and EK certificates that they plan to pre-provision.

Note:

- PTP 1.03 required support for RSA 2k and ECC P256 EK Certificates.
- PTP 1.04 required support for RSA 2k and ECC P384 EK Certificates.
- PTP 1.05 required support for RSA 2k and ECC P384 EK Certificates and allowed RSA 3k EK Certificates.

A TPM vendor is not required to provide an RSA 3072-bit EK Certificate if they provide an ECC P384 EK Certificate.

End of informative comment

1. If a TPM is preprovisioned with EK Certificates, it SHALL be provisioned with EK Certificates for an RSA EK and an ECC EK.

Table 6 — EK Certificate Allowed Combinations

	ECC_NIST_P256	ECC_NIST_P384
RSA 2048		A
RSA 3072	B	C

- a. A conformant TPM SHALL preprovision at least EK Certificates with the algorithm combinations of A or B or C from Table 6 — EK Certificate Allowed Combinations. A conformant TPM MAY preprovision additional EK Certificates.
- b. All Certificates SHALL comply with the definition in the TCG EK Credential Profile for TPM Family 2.0 [14].
2. The index SHALL be defined as an ordinary index.
3. NV Index attributes TPMA_NV_AUTHWRITE, TPMA_NV_OWNERWRITE, TPMA_NV_GLOBALLOCK, TPMA_NV_CLEAR_STCLEAR, TPMA_NV_ORDERLY and TPMA_NV_READ_STCLEAR SHALL be CLEAR.
4. NV Index attributes TPMA_NV_POLICY_DELETE and TPMA_NV_POLICYWRITE SHOULD be CLEAR.
5. NV Index attributes TPMA_NV_PLATFORMCREATE, TPMA_NV_AUTHREAD, TPMA_NV_OWNERREAD and TPMA_NV_NO_DA SHALL be SET.
6. NV Index attribute TPMA_NV_PPWRITE SHOULD be SET.
7. All other NV Index attributes MAY be SET.
8. The authorization value for the EK certificate index SHALL be an Empty Auth as defined in a TPM 2.0 Library Specification, Part 1, Terms and Abbreviations.
9. The authorization policy for the EK certificate index SHOULD be an Empty Buffer as defined in a TPM 2.0 Library Specification, Part 1, Terms and Abbreviations.
10. The hash algorithm used for the EK certificate signature SHALL be of equal or better security strength as the key used to sign the certificate.

4.5.4 TPM NV Indices for Counters and PINs

Start of informative comment

The following requirements define the minimum number of NV Indices which should be configurable for counter and PIN functionality, at point of delivery to a platform manufacturer. If NV Indices have already been defined (in platform manufacturing or in normal use), a TPM may not support the full number of indices for Counters and PINs as defined below. A TPM may report a value of 0 in TPM_PT_NV_COUNTERS_MAX if there is no fixed maximum. The number of counter indexes is determined by the available NV memory pool.

End of informative comment

1. A TPM SHALL support a minimum of 6 NV Indices with the attribute TPM_NT_COUNTER. Additionally, two of these counters SHALL support TPMA_NV_ORDERLY SET. Note: This requirement applies even if there are NV indexes created in an EPI memory device. See Section 4.5.4.2 Extended Peripheral Interface
2. A TPM SHALL support a minimum of 2 NV Indices with either the attribute TPM_NT_PIN_FAIL or TPM_NT_PIN_PASS.

4.5.5 General Purpose I/O (GPIO)**Start of informative comment**

General purpose I/O (GPIO) provides an optional interface between a TPM's command interface and an external device. The actual use and protocol of the signal is implementation specific and is not specified by TCG.

A TPM's command interface accesses the GPIO pins using the NV Storage interface. This is much like memory-mapped I/O in other architectures. This specification only defines the routing of the GPIO index to the GPIO pin.

Because GPIO can be used for security or privacy functions, it must not be open, by default, for public access. For this reason, it is required that the NV Storage area that is mapped to the GPIO be "defined" like any other NV Storage area prior to allowing its use. GPIO indices may be defined by a platform manufacturer or by the owner of a system. The system owner would act as a TPM Owner. When defining the index, the Owner may elect to assign access rights per the normal attributes. If a TPM Owner is removed, the area returns to undefined and must be defined again before use. The reason for this behavior is that the new TPM Owner may have different security and privacy requirements for this GPIO. If defined by the platform manufacturer, the index would be expected to remain even in the presence of a new owner. An example where this might be used would be in a system where the GPIO is used to control an indicator when certain conditions attested to by a TPM are true.

The range reserved for GPIO is not specific to a platform. It is, therefore, a requirement that Software or other platform processes using GPIOs understand the nature of the platform before using it (i.e., which NV Storage Index is associated with which GPIO and the purpose of the GPIO on that platform).

Note that the pin-out specified in Section 9.1 TPM Packaging is only recommended and is not mandatory. TPMs can be implemented using any packaging. However, if this packaging is chosen, the pins, including the location of the GPIO pins, are mandatory. If this packaging is not used, a TPM manufacturer must provide documentation to the platform manufacturer indicating which pin is used as a GPIO pin.

End of informative comment

Implementation of this section is optional; but if implemented, it MUST be done in the manner specified in this section.

4.5.5.1 Reserved NV Storage Indices for GPIO**Start of informative comment**

The following reserved NV handles are normatively assigned by TCG in the Registry of Reserved TPM Handles and Localities [13].

End of informative comment

If implemented, GPIO pins SHALL be mapped to NV Indices in the range of 0x01C40000 to 0x01C4000F, one index per one GPIO.

4.5.5.2 Extended Peripheral Interface**Start of informative comment**

The Extended Peripheral Interface (EPI) is an optional feature that provides the capability to connect a serial peripheral device, based on I2C or SPI bus interfaces, to a TPM's GPIO pins. A TPM serves as an authenticator for any access to this downstream peripheral by means of a TPM's NV commands. The data sent to and received from the downstream peripheral is opaque to a TPM.

Regardless of the type of peripheral device attached to the EPI, a configuration index is required to use this feature. The configuration index defines how a TPM should set up the communication channel to the peripheral device, and, if it is a memory device, the size of the memory device in bytes. A group of NV Index handles has been reserved in the TCG Registry of Reserved Handles and NV Indices. It is expected that the configuration index will be defined during platform manufacturing and would be defined using the first reserved handle. However, this is not a firm requirement. If a peripheral device is something other than a memory device, then it is feasible that only one data buffer index would be defined, using one of the additional handles reserved for this purpose. If the peripheral device is a memory device, then multiple data buffer indices could be defined, on a first come, first served basis with different owners.

The configuration index fields are defined in this specification so that vendors may define proprietary communication channel parameters, but memory devices can be used by multiple entities. The first generation of the EPI did not provide for confidentiality or integrity protection of the data written to the peripheral device, but as of PTP 1.06 such protection is provided. Therefore, a version field is included in the configuration index.

The configuration indexes are defined and written by the platform manufacturer, as enablement of this feature requires the platform manufacturer to connect an external device, e.g., a flash memory IC, to a TPM GPIO pins. Data buffer indexes can be created as part of TPM provisioning by the platform manufacturer or can be created post-provisioning by an OS or application.

If an external memory device is attached to the EPI, some types of NV indexes, such as NV counters, might be incompatible with external memory. That is because, when a new counter is created a TPM has to initialize the counter with the maximum counter value already in use (MAX_COUNTER).

As of PTP 1.06, a TPM can optionally provide rollback protection for data written to the peripheral memory device. This PTP does not normatively define an implementation. One example of the method a vendor might use leverages a multi-level organization of rollback counters (EPI-count values) and is described below:

1. Each EPI-NV Index has a EPI-count value that starts at zero and increments every time the index changes.
2. An array of EPI-count values for each writeable segment is stored in the external memory at level 1. The array also has an EPI-count value.
3. An array of level 1 EPI-count values is stored in level 2.
4. The final EPI-count value is stored inside a TPM.

Take an example with multiple NV Indices; each index has an EPI-count. An EPI-Counter File contains the EPI-count values from each NV Index. A TPM stores an EPI-count, which is also saved in the EPI-Counter File. If any index is changed, its EPI-count value increments, the corresponding EPI-count value in the EPI-Counter file is incremented, a TPM EPI-count value in the EPI-Counter file is updated and the EPI-count stored in a TPM is updated.

Assuming a 4-byte count and a 1k TPM EPI-Counter array size, a TPM EPI-count value can support an array of 256 counts. The 1k TPM EPI-Counter array divided by 4 results in 256. Each layer with a 4-byte EPI-Count exponentially increases the number of counters. One level supports in 256 counters, two levels supports 64k counters, three levels supports 16M counters, etc.

Each read traverses the levels to ensure no rollback. Each write, actually a read-modify-write, also traverses the levels to write the tree and finally a TPM single count value.

End of informative comment

Requirements for EPI Configuration Index

1. If a TPM implements an EPI a TPM SHALL support creation of a configuration index with the format defined in Table 7 — EPI Configuration Index Definition:

Table 7 — EPI Configuration Index Definition

Field	M/O	Offset (decimal)	Size	Value and Description
Version	M	0	1B	Version of the Config structure, SHALL be 02h
EPI Bus Configuration	M	1	4B	Vendor specific configuration information that defines the parameters of the physical interface to the peripheral device
MetaDataSize	O1	5	2B	Size of the Memory Device Meta Data area
Memory Device Size	O1	7	4B	If the peripheral device is a memory device, the value of this field SHALL be the total addressable size in KB (1024 Bytes) of the attached memory device.
Memory Device Meta Data	O1	11	MetaData Size	Information about the type of memory device, such as type of flash, block size, etc. Vendor specific

2. The EPI Configuration Index MAY be defined with a handle type for the NV Index equal to 0x11 (TPM_HT_EXTERNAL_NV).
3. Extended Peripheral Interface Configuration Indices SHALL be defined with the attribute TPM_NT_ORDINARY.
4. The EPI Configuration index, if predefined by a TPM manufacturer, SHALL be defined as an ordinary index with the following attributes:
 - a. NV Index attributes TPMA_NV_AUTHWRITE, TPMA_NV_OWNERWRITE, TPMA_NV_GLOBALLOCK, TPMA_NV_CLEAR_STCLEAR, TPMA_NV_ORDERLY and TPMA_NV_READ_STCLEAR SHALL be CLEAR.
 - b. NV Index attributes TPMA_NV_POLICY_DELETE and TPMA_NV_POLICYWRITE SHOULD be CLEAR.
 - c. NV Index attributes TPMA_NV_PLATFORMCREATE, TPMA_NV_AUTHREAD, TPMA_NV_OWNERREAD and TPMA_NV_NO_DA SHALL be SET.
 - d. NV Index attribute TPMA_NV_PPWRITE SHOULD be SET.
 - e. All other NV Index attributes MAY be SET.

General Requirements for the EPI Indexes

1. All the following requirements MUST be implemented if a TPM supports the EPI feature.
 - a. A TPM SHALL support definition of Extended Peripheral Interface NV Indices, with the handle type for any EPI-NV Index equal to TPM_HT_EPI_INDEX.
 - b. A TPM SHALL map TPM2_NV* commands sent to these indices to the Extended Peripheral Interface. **Note:** All TPM2_NV* commands behave as defined in a TPM2 Library Specification unless otherwise defined in this section.
 - c. A TPM SHALL return a response to a TPM2_NV_Write command only after it has completed the write to the downstream device.
 - d. A TPM SHALL return a response to a TPM2_NV_Read command only after receiving all the data from the downstream device.

Requirements when External Device is not a Memory Device

Start of informative comment

The EPI can be used to as an analog to a TPM 1.2 with GPIO pins. Connecting a door lock or a smart card reader to a TPM provides access control to pass data between the caller and the external device.

End of informative comment

1. Extended Peripheral Interface NV Indexes SHALL be defined with the attribute TPMA_NV_ORDERLY CLEAR and the attributes TPMA_NV_PLATFORMCREATE SET.
2. EPI Data Indexes SHALL be defined to have the attribute TPM_NT_ORDINARY.
3. On receipt of a TPM2_NV_DefineSpace command to define an EPI Data Index in the EPI NV Index range, a TPM SHALL map the requested EPI-NV Index to a TPM's GPIO pin.

Requirements when External Device is a Memory Device

1. The EPI Configuration Memory Device Size field SHALL be non-zero.
2. On receipt of a TPM2_NV_DefineSpace command to define an EPI data buffer in the EPI NV Index range, a TPM SHALL map the requested portion of the Memory Device.
3. A TPM SHALL NOT restrict creation of additional EPI data buffer indices unless a TPM has insufficient NV space to support creation of the index attributes, or the EPI memory device has insufficient space for the requested size of the index.
4. A TPM MAY support creation of an EPI-NV index with a TPMS_NV_PUBLIC and TPM2B_AUTH stored in the external memory device.
5. A TPM SHALL provide integrity protection for all data written to the external memory device.
6. A TPM SHALL provide confidentiality protection for all data written to the external memory device, except TPMS_NV_PUBLIC data.
7. If a TPM detects an integrity error in data written to the external memory, a TPM SHALL return TPM_RC_INTEGRITY.
8. A TPM MAY provide rollback detection.
 - a. If a TPM detects a rollback attempt, a TPM SHALL return TPM_RC_INTEGRITY.
9. A TPM2_NV_DefineSpace command with the attributes in the Support Required column of Table 8 — TPMA_NV Support Requirements for EPI Memory Devices and Table 9 — TPM_NT Support for EPI Memory Devices listed as yes SHALL support definition of the index with that attribute.
 - a. Attributes listed as No MAY be supported.
 - b. If a TPM does not support an attribute, TPM SHALL return TPM_RC_ATTRIBUTES if the attribute is SET.
10. If a TPM detects a missing EPI memory device, the TPM SHOULD return TPM_RC_NV_UNAVAILABLE.

Table 8 — TPMA_NV Support Requirements for EPI Memory Devices

Bit	Name	Support Required for an EPI Memory Device
0	TPMA_NV_PPWRITE	Yes
1	TPMA_NV_OWNERWRITE	Yes
2	TPMA_NV_AUTHWRITE	Yes
3	TPMA_NV_POLICYWRITE	Yes
7:4	TPM_NT	See Table 9 — TPM_NT Support for EPI Memory Devices
9:8	Reserved	Yes
10	TPMA_NV_POLICY_DELETE	Yes
11	TPMA_NV_WRITELOCKED	N/A
12	TPMA_NV_WRITEALL	Yes
13	TPMA_NV_WRITEDEFINE	Yes
14	TPMA_NV_WRITE_STCLEAR	Yes
15	TPMA_NV_GLOBALLOCK	No
16	TPMA_NV_PPREAD	Yes
17	TPMA_NV_OWNERREAD	Yes
18	TPMA_NV_AUTHREAD	Yes
19	TPMA_NV_POLICYREAD	Yes
24:20	Reserved	Yes
25	TPMA_NV_NO_DA	Yes
26	TPMA_NV_ORDERLY	No
27	TPMA_NV_CLEAR_STCLEAR	No
28	TPMA_NV_READLOCKED	N/A
29	TPMA_NV_WRITTEN	N/A
30	TPMA_NV_PLATFORMCREATE	Yes
31	TPMA_NV_READ_STCLEAR	No

Table 9 — TPM_NT Support for EPI Memory Devices

Name	Value	Description	Support Required for an EPI Index
TPM_NT_ORDINARY	0x0	Ordinary – contains data that is opaque to a TPM that can only be modified using TPM2_NV_Write().	Yes
TPM_NT_COUNTER	0x1	Counter – contains an 8-octet value that is to be used as a counter and can only be modified with TPM2_NV_Increment().	No
TPM_NT_BITS	0x2	Bit Field – contains an 8-octet value to be used as a bit field and can only be modified with TPM2_NV_SetBits().	No
TPM_NT_EXTEND	0x4	Extend – contains a digest-sized value used like a PCR. The Index can only be modified using TPM2_NV_Extend(). The extend will use the nameAlg of the Index.	No
TPM_NT_PIN_FAIL	0x8	PIN Fail - contains <i>pinCount</i> that increments on a PIN authorization failure and a <i>pinLimit</i>	No
TPM_NT_PIN_PASS	0x9	PIN Pass - contains <i>pinCount</i> that increments on a PIN authorization success and a <i>pinLimit</i>	No

4.6 PCR Requirements

Start of informative comment

This section specifies the number and attributes of the set of PCRs required for a PC Client Platform. The purpose for specifying this is to establish common and expected behavior for both platform hardware and Software.

A TPM 2.0 Library Specification allows TPM vendors to implement PCR in NV. Software which performs multiple extends to PCRs in a boot cycle could subject a TPM PCR to NV wear-out. It is therefore recommended to use RAM for PCRs. If a TPM uses NV for PCR then the vendor is strongly recommended to provide a cache for the most recently used PCRs.

A TPM2_PCR_Allocate command is used to assign HASH algorithms for PCR banks. It is also used to allocate PCR within a bank. The PC Client Specific Platform Firmware Profile for TPM 2.0 Systems [3] defines the requirements for a platform manufacturer to allocate the necessary number of PCR for a PC Client TPM. It is not expected that a TPM vendor will produce a TPM that returns an error code if platform firmware does not allocate the TCG PC Client Platform Firmware Profile [3] defined number of PCR.

End of informative comment

1. A conformant TPM SHALL allow an allocation of a minimum of 24 PCRs, 0-23, within all allocated banks. A conformant TPM MAY support more than one bank of PCRs.
2. A conformant TPM MAY support allocation of fewer than PCR 0-23 in any bank, or a TPM MAY return TPM_RC_NO_RESULT in response to a TPM2_PCR_ALLOCATE command.
3. A conformant TPM SHALL support SHA-384 (0x000C) and SHA-256 (0x000B). A TPM MAY support additional Hash algorithms.
 - a. If a TPM supports only one bank of PCRs,
 - i. The default Hash Algorithm ID for the PCR SHALL be 0x000B (SHA-256).
 - ii. A TPM SHALL support allocation of this bank for any supported Hash algorithm.
 - b. If a TPM supports multiple banks of PCRs:

- i. A TPM SHALL enable by default the required Hash algorithms as specified in normative 3.
 - ii. If a TPM supports additional Hash algorithms, a TPM SHALL support PCR 0-23 within each allocated bank for any combination of allocated algorithms for the supported banks of PCRs. Example: If a TPM supports five Hash algorithms, and supports two banks of PCRs, each of the two PCR banks can be assigned any one of the five algorithms.
4. If a TPM is implemented with more than 24 PCRs in a bank, the attributes of the additional PCRs are not defined by this specification.
 5. A conformant TPM SHALL configure an Empty Auth as the authorization value for all PCRs.
 6. A conformant TPM SHALL configure an Empty Policy as the authorization policy for all PCRs.
 7. The optional TPM2_PCR_SetAuthPolicy and TPM2_PCR_SetAuthValue commands, if implemented, SHALL return TPM_RC_VALUE.
 8. For this specification, the D-RTM PCR SHALL be PCR 17 and the S-HCRTM PCR SHALL be PCR 0.

4.6.1 PCR Attributes

Start of informative comment

PC Client TPM PCRs are defined to enable a PC Client TPM to support D-RTM. See Section 5.3.1 D-RTM Execution Sequence.

Note that since the hardware that performs the DRTM sequence at Locality 4 is incapable of doing TPM2_PCR_Reset, a TPM_PT_PCR_RESET_L4 attribute is repurposed to indicate the initial state of the PCR (0 or -1) and to indicate which PCR are set to 0 by a successful D-RTM Sequence.

The attributes defined in Table 10 are specified in a TPM Library Specification Part 2 [7]. The attribute TPM_PT_PCR_SAVE defines the behavior of a TPM after TPM2_Shutdown (STATE) followed by a TPM2_Startup (STATE) (TPM Resume).

End of informative comment

1. For a PC Client TPM, a value in the “Reset by TPM2_PCR_Reset for Locality = x” column (column 8) in Table 10 of:
 - a. N (No): means that a TPM2_PCR_Reset command SHALL NOT reset the indicated PCR.
 - b. Y (Yes): means that a TPM2_PCR_Reset command SHALL reset the indicated PCR.
2. For a PC Client TPM, a value in the “Reset by D-RTM Event Locality = x” column (column 7) in Table 10 of:
 - a. N (No): means that the D-RTM Sequence SHALL NOT reset the indicated PCR,
 - b. Y(Yes): means that the D-RTM Sequence SHALL reset the indicated PCR.
3. For a PC Client TPM, for each PCR, the value in the “Extended by TPM2_PCR_Extend Locality = x” column (Column 9) in Table 10 of:
 - a. N (No): means that a TPM2_PCR_Extend or TPM2_PCR_Event command SHALL NOT extend the indicated PCR.
 - b. Y (Yes): means that a TPM2_PCR_Extend or TPM2_PCR_Event command SHALL extend the indicated PCR.
4. For a PC Client TPM, a TPM SHALL return the values defined in the “Value of TPM_PT_PCR_RESET_LX” column (Table 10, column 10) in response to a TPM2_GetCapability command.
5. The initialization value for each PCR is defined in Section 4.6.2, Table 11.

Table 10 — PCR Attributes

1	2	3	4	5	6	7	8	9	10
PCR Index	Alias	TPM_PT_PCR_SAVE	TPM_PT_PCR_AUTH	TPM_PT_PCR_POLICY	TPM_PT_PCR_NO_INCREMENT	Reset by D-RTM Event Locality = x X=4,3,2,1,0	Reset by TPM2_PCR_Reset Locality = x X=4,3,2,1,0	Extended by TPM2_PCR_Extend Locality = x X= 4,3,2,1,0	Value of TPM_PT_PCR_RESET_Lx Locality = x X= 4,3,2,1,0
0 – 15	Static RTM	1	0	0	0	N,N,N,N,N	N,N,N,N,N	Y,Y,Y,Y,Y	0,0,0,0,0
16	Debug	0	0	0	1	N,N,N,N,N	N,Y,Y,Y,Y	Y,Y,Y,Y,Y	0,1,1,1,1
17	Locality 4	0	0	0	0	Y,N,N,N,N	N,N,N,N,N	Y,Y,Y,N,N	1,0,0,0,0
18	Locality 3	0	0	0	0	Y,N,N,N,N	N,N,N,N,N	Y,Y,Y,N,N	1,0,0,0,0
19	Locality 2	0	0	0	0	Y,N,N,N,N	N,N,N,N,N	N,Y,Y,N,N	1,0,0,0,0
20	Locality 1	0	0	0	0	Y,N,N,N,N	N,Y,Y,N,N	N,Y,Y,Y,N	1,1,1,0,0
21	Dynamic OS Controlled	0	0	0	1	Y,N,N,N,N	N,Y,Y,N,N	N,N,Y,N,N	1,1,1,0,0
22	Dynamic OS Controlled	0	0	0	1	Y,N,N,N,N	N,Y,Y,N,N	N,N,Y,N,N	1,1,1,0,0
23	Application Specific	0	0	0	1	N,N,N,N,N	N,Y,Y,Y,Y	Y,Y,Y,Y,Y	0,1,1,1,1

4.6.2 PCR Initial and Reset Values

Start of informative comment

The contents of the cells in Table 11 are the values to which each of the PCRs is initialized prior to being transformed by the command or sequence called out in the title of each column. The actual transformation is defined in a TPM Library Specification [7].

Within the context of specifying the Reset Value for PCRs, the value -1 is defined to be the same size, in bytes, of the digest for the supported Hash Algorithm ID with all bits set to the value of 1.

The column “No S-HCRTM Sequence” indicates that no S-HCRTM sequence was initiated prior to a TPM receiving a TPM2_Startup (CLEAR) indication. The column “S-HCRTM Sequence” indicates that an S-HCRTM sequence was initiated prior to a TPM receiving a TPM2_Startup (CLEAR) indication.

End of informative comment

Table 11 — PCR Initial and Reset Values

	TPM2_Startup(CLEAR)		TPM2_PCR_Reset
--	---------------------	--	----------------

PCR Index	No S-HCRTM Sequence	S-HCRTM Sequence	HASH_END (DRTM Sequence)	
0	Locality Indicator ¹	Updated per S-HCRTM sequence ²	NC	NC
1-15	0	0	NC	NC
16	0	0	NC	0
17	-1	-1	Updated per D-RTM sequence	NC
18-19	-1	-1	0	NC
20-22	-1	-1	0	0
23	0	0	NC	0

Note 1: Locality Indicator is locality at which TPM2_Startup(CLEAR) is received.
Note 2: See Section 5.3.2 S-HCRTM Execution Sequence.
 NC = No Change

4.7 Power Management

Start of informative comment

While power management of a device is allowed by the standard which specifies the LPC interface (if implemented by a TPM), a TPM is designed to be either fully functional (ACPI defined device power management state D0) or not functional (ACPI defined device power management state D3). In practical applications of TPM, power management of a TPM has no real meaning. The TCG specifications define TPM behavior and functions to simplify a TPM's interactions with the platform's components including the Software. A TPM2_Shutdown (STATE) and TPM2_Startup commands were created as a mechanism for the platform's Software and BIOS to communicate entry into and exit from the D3 Power State. A TPM2_Shutdown (STATE) command allows a Static OS to indicate to a TPM that the platform may enter a low power state where a TPM will be required to enter the D3 power state. The use of the term "may" is significant in that there is no requirement for the platform to enter the low power state after sending a TPM2_Shutdown (STATE) command. The Software may, in fact, send subsequent commands after sending a TPM2_Shutdown (STATE) commands. A TPM2_Shutdown (STATE) command simply tells a TPM to save the required volatile contents because power to a TPM may be removed at any time. A TPM is responsible for tracking its internal state so that, if a command that alters a TPM's saved state is sent to a TPM after a TPM2_Shutdown (STATE) command, a TPM voids the saved internal state so a subsequent TPM2_Startup(STATE) will fail. In this case, it is the responsibility of platform Software to send a subsequent TPM2_Shutdown (STATE) command to preserve the new internal state of a TPM.

It is the responsibility of the S-CRTM to indicate to a TPM using a TPM2_Startup command whether a TPM must reset or restore its saved state (e.g., PCR values, etc.). If the S-CRTM commands a TPM to restore the saved state (i.e., STATE), this restores the transitive trust chain. If the S-CRTM commands a TPM to reset the saved state (i.e., CLEAR), this clears and restarts a new transitive trust state. The rationale here is that the S-CRTM is trusted to establish the initial transitive trust chain, so it should also be trusted to determine whether to restore or clear it.

Power management has changed since the original LPC specification and TPM Interface Specification were produced. The LPCPD# pin, as defined in the LPC specification, is a shared pin allowing for a power management protocol for ACPI S3-aware devices on the LPC bus. As TPMs do not know or participate in Suspend to RAM (ACPI S3), this pin has no meaning for a TPM. As such, the implementation of the LPCPD# pin on a TPM is platform and chipset implementation specific. If TPM vendors implement the LPCPD# pin and power management protocol, they should provide documentation indicating the method to disable the function.

In the PC Client TPM Interface Specification 1.3 [15], the concept of a lower power operating mode was introduced which allows a TPM to enter a lower power state under specific considerations. A TPM, if in the idle state, can reduce its power consumption by shutting down internal functional blocks if the SPI or LPC interface and a TPM registers remain active. The intention is to prevent any impact to existing TPM drivers. When a TPM receives a transaction on its interface that would cause it to move from Idle to Ready, a TPM must exit the low power mode within TIMEOUT_B, see Section 6.5.1.4. There is no additional signaling or register bits required to transition a TPM into or

out of a low power state. Because of the performance limitations of the pre-boot environment, this specification does not allow a TPM to enter a low power state prior to the receipt of a TPM2_Startup command.

End of informative comment

1. After _TPM_INIT, a TPM SHALL behave as if it is in ACPI Device Power State D0 even if it supports ACPI Device Power States D1-D2.
2. A TPM SHALL NOT accept commands unless it is in the ACPI Device Power State D0.
3. A TPM SHALL NOT exit the ACPI Device Power State D3 unless it receives _TPM_INIT.
4. A TPM SHALL NOT enter an alternative ACPI Device Power State upon receipt of a TPM2_Shutdown (State) command.
5. If implementing an LPC TPM, a TPM SHALL be implemented to allow for the LPC power management protocol to be disabled by strapping LPCPD# pin HIGH.
6. If implementing an SPI TPM, a TPM MAY support lower power states ONLY if a TPM is in the Idle state.
 - a. If lower power states are supported, a TPM SHALL respond to requests to transition to the Ready state within TIMEOUT_B.
 - b. A TPM SHALL NOT enter any lower power state between receipt of _TPM_INIT and receipt of a TPM2_Startup command.

4.8 Self-Test Requirements

Start of informative comment

A TPM 2.0 Library Specification has three ways for a TPM to test functions and capabilities: TPM2_SelfTest, TPM2_IncrementalTest, and on-demand testing.

The command TPM2_SelfTest provides a flag (fullTest) to allow a caller to control whether a TPM performs a full self-test or a partial self-test. TPM2_IncrementalTest provides a means to specify which capabilities should be tested. On-demand testing allows a TPM to test an untested capability when it is invoked.

To make TPM behavior more deterministic for PC Client platforms, this specification constrains the behavior for TPM2_SelfTest as compared to a TPM Library Specification [7]. With the fullTest flag set to YES, a TPM will perform testing so that it mirrors the behavior of a TPM 1.2 TPM_SelfTestFull command. With the fullTest flag set to NO, a TPM will test as specified in the following normative text so that its behavior is analogous to a TPM 1.2 TPM_ContinueSelfTest command.

This specification does not constrain TPM2_IncrementalTest or on-demand testing.

End of informative comment

1. On receipt of TPM2_SelfTest(fullTest == NO), if any test is required, a TPM SHALL return TPM_RC_TESTING and perform background testing of untested functions. If all required testing has been performed, a TPM SHALL return TPM_RC_SUCCESS.

Note: The test status can be retrieved from a TPM using TPM2_GetTestResult.

2. On receipt of TPM2_SelfTest(fullTest == YES), a TPM SHALL perform a full self-test and return the result when all tests are complete.

4.9 Firmware Upgrade

Start of informative comment

A TPM 2.0 Library Specification provides a standardized mechanism for upgrading a TPM's firmware. A TPM compliant to this specification must implement a firmware upgrade mechanism but is not required to implement the firmware upgrade specified in a TPM 2.0 Library Specification.

End of informative comment

1. A TPM compliant to this specification SHALL implement firmware upgrade.

2. A TPM compliant to this specification MAY implement a TPM 2.0 Library Specification defined firmware upgrade.
3. If a TPM implements the Library-defined firmware upgrade, a TPM SHALL implement the commands TPM2_FieldUpgrade_Start and TPM2_FieldUpgrade_Data, and MAY implement TPM2_Firmware_Read.

DRAFT

5 TPM Capabilities and Commands

5.1 Platform Specific Requirements

Start of informative comment

This section contains miscellaneous items that a TPM 2.0 Library Specification recommends Platform Work Groups define or constrain in the Platform Specific Profiles.

End of informative comment

5.1.1 TPM Handles, Objects and Contexts

1. A TPM SHALL implement an 8-byte objectContextID counter at a minimum.
2. A TPM SHALL implement a mechanism to reuse Object Handles.
3. A TPM SHALL NOT return TPM_RC_OBJECT_HANDLES.

5.1.2 Authenticated Countdown Timer (ACT)

If a TPM implements the optional TPM2_ACT_SetTimeout command:

1. A TPM SHALL support one ACT instance that MAY assert a GPIO pin when the ACT times out.
2. The ACT SHALL advance when a TPM is in any device power state other than D3.

Note: This specification does not define the configuration of the GPIO pin. The pin may be attached to a system signal that causes a platform action such as a platform reset, when asserted. TPM vendors should provide guidance to platform implementers regarding configuration of this pin to avoid unintended results. At _TPM_INIT, a TPM can sample this pin to determine the non-assertion state. If the ACT times out, a TPM will actively drive the GPIO pin to the assert state to trigger the desired system action.

5.2 Command Table

Start of informative comment

A TPM 2.0 Library Specification defines the Protected Capabilities (commands) for many types of platforms in a manner that is not platform specific. Not all the Protected Capabilities in a TPM 2.0 Library Specification are applicable to all platforms, and it is left to the platform specific specifications to enumerate which of the commands are required for TPMs meant to be used in that type of platform.

Any commands implemented in a newer (later than version 1.59) version of the TPM Library specification not listed in Table 12 — Mandatory/Optional TPM Commands are considered optional.

End of informative comment

1. To be compliant with this specification, a TPM SHALL support the commands labeled as mandatory (M) in the column labeled “M / O” in Table 12.

Table 12 — Mandatory/Optional TPM Commands

Commands	M / O	Comments
Signals / Indications		
_TPM_INIT	M	
_TPM_Hash_Start	M	
_TPM_Hash_Data	M	
_TPM_Hash_End	M	

Commands	M / O	Comments
Startup		
TPM2_Startup	M	
TPM2_Shutdown	M	
Testing		
TPM2_IncrementalSelfTest	M	
TPM2_SelfTest	M	
TPM2_GetTestResult	M	
Session Commands		
TPM2_StartAuthSession	M	
TPM2_PolicyRestart	M	
Object Commands		
TPM2_Create	M	
TPM2_Load	M	
TPM2_LoadExternal	M	
TPM2_ReadPublic	M	
TPM2_ActivateCredential	M	
TPM2_MakeCredential	M	
TPM2_Unseal	M	
TPM2_ObjectChangeAuth	M	
TPM2_CreateLoaded	O	Made optional in PTP 1.06
Duplicate Commands		
TPM2_Duplicate	M	
TPM2_Rewrap	O	
TPM2_Import	M	
Asymmetric Primitives		
TPM2_RSA_Encrypt	M	
TPM2_RSA_Decrypt	M	
TPM2_ECDH_KeyGen	M	
TPM2_ECDH_ZGen	M	
TPM2_ECC_Parameters	M	
TPM2_ZGen_2Phase	O	

Commands	M / O	Comments
Symmetric Primitives		
TPM2_EncryptDecrypt	O	This command was deprecated in TPM Library Specification version 1.38 and replaced by TPM2_EncryptDecrypt2.
TPM2_EncryptDecrypt2	O	
TPM2_Hash	M	
TPM2_HMAC	M	
TPM2_MAC	O	Added in PTP 1.05. This command uses the same ordinal as TPM2_HMAC. If implemented, selection between TPM2_HMAC and TPM2_MAC is done via parameters, as defined in a TPM Library Specification.
Random Number Generator		
TPM2_GetRandom	M	
TPM2_StirRandom	M	
Hash/HMAC/Event Sequences		
TPM2_HMAC_Start	M	
TPM2_MAC_Start	O	Added in PTP 1.05. This command uses the same ordinal as TPM2_HMAC_Start. If implemented, selection between TPM2_HMAC_Start and TPM2_MAC_Start is done via parameters, as defined in a TPM Library Specification.
TPM2_HashSequenceStart	M	
TPM2_SequenceUpdate	M	
TPM2_SequenceComplete	M	
TPM2_EventSequenceComplete	M	
Attestation Commands		
TPM2_Certify	M	
TPM2_CertifyCreation	M	
TPM2_Quote	M	
TPM2_GetSessionAuditDigest	M	
TPM2_GetCommandAuditDigest	O	
TPM2_GetTime	M	Added in PTP 1.03
TPM2_CertifyX509	O	

Commands	M / O	Comments
Anonymous Attestation		
TPM2_Commit	O	Made optional in PTP 1.05
TPM2_EC_Ephemeral	O	
Signature Verification		
TPM2_VerifySignature	M	
TPM2_Sign	M	
Command Audit		
TPM2_SetCommandCodeAuditStatus	O	
Integrity Collection (PCR)		
TPM2_PCR_Extend	M	
TPM2_PCR_Event	M	
TPM2_PCR_Read	M	
TPM2_PCR_Allocate	M	
TPM2_PCR_SetAuthPolicy	O	This command has no meaning for a PCR as PC Client TPMs do not implement authorization policy for PCR ¹
TPM2_PCR_SetAuthValue	O	This command has no meaning for a PCR as PC Client TPMs do not implement authorization for PCR
TPM2_PCR_Reset	M	

¹ See Table 10 TPM_PT_PCR_POLICY attributes for all mandatory PCR.

Commands	M / O	Comments
Enhanced Authorization (EA)		
TPM2_PolicySigned	M	
TPM2_PolicySecret	M	
TPM2_PolicyTicket	M	
TPM2_PolicyOR	M	
TPM2_PolicyPCR	M	
TPM2_PolicyLocality	M	
TPM2_PolicyNV	M	
TPM2_PolicyCounterTimer	M	
TPM2_PolicyCommandCode	M	
TPM2_PolicyPhysicalPresence	O	Required if a TPM implements Physical Presence
TPM2_PolicyCpHash	M	
TPM2_PolicyNameHash	M	
TPM2_PolicyDuplicationSelect	M	
TPM2_PolicyAuthorize	M	
TPM2_PolicyAuthValue	M	
TPM2_PolicyPassword	M	
TPM2_PolicyGetDigest	M	
TPM2_PolicyNvWritten	M	
TPM2_PolicyTemplate	M	Added in PTP 1.04
TPM2_PolicyAuthorizeNV	M	Added in PTP 1.04
Hierarchy Commands		
TPM2_CreatePrimary	M	
TPM2_HierarchyControl	M	
TPM2_SetPrimaryPolicy	M	
TPM2_ChangePPS	O	This command may be required for successful completion of a FIPS140 evaluation
TPM2_ChangeEPS	O	This command may be required for successful completion of a FIPS140 evaluation
TPM2_Clear	M	
TPM2_ClearControl	M	
TPM2_HierarchyChangeAuth	M	

Commands	M / O	Comments
Dictionary Attack Functions		
TPM2_DictionaryAttackLockReset	M	
TPM2_DictionaryAttackParameters	M	
Miscellaneous Management Functions		
TPM2_PP_Commands	O	
TPM2_SetAlgorithmSet	O	
Field Upgrade		
TPM2_FieldUpgradeStart	O	Both commands are required if either is implemented
TPM2_FieldUpgradeData		
TPM2_FirmwareRead	O	
Context Management		
TPM2_ContextSave	M	
TPM2_ContextLoad	M	
TPM2_FlushContext	M	
TPM2_EvictControl	M	
Clocks and Timers		
TPM2_ReadClock	M	
TPM2_ClockSet	M	
TPM2_ClockRateAdjust	M	
Capability Commands		
TPM2_GetCapability	M	
TPM2_TestParms	M	

Commands	M / O	Comments
Non-volatile Storage		
TPM2_NV_DefineSpace	M	
TPM2_NV_UndefineSpace	M	
TPM2_NV_UndefineSpaceSpecial	M	
TPM2_NV_ReadPublic	M	
TPM2_NV_Write	M	
TPM2_NV_Increment	M	
TPM2_NV_Extend	M	
TPM2_NV_SetBits	M	
TPM2_NV_WriteLock	M	
TPM2_NV_GlobalWriteLock	O	
TPM2_NV_Read	M	
TPM2_NV_ReadLock	M	
TPM2_NV_ChangeAuth	M	
TPM2_NV_Certify	M	
Attached Component Commands		
TPM2_AC_GetCapability	O	Added in PTP 1.05. If implemented, all three commands are required.
TPM2_AC_Send		
TPM2_Policy_AC_Send		
Authenticated Countdown Timer		
TPM2_ACT_SetTimeout	O	Added in PTP 1.05

5.3 Locality-Controlled Functions

5.3.1 D-RTM Execution Sequence

Start of informative comment

The D-RTM is started while the platform may be in an untrusted state. Special trusted mechanisms must be established to communicate the source of the corresponding commands to a TPM. These commands are indicated and controlled by the appropriate locality.

Locality 4 has the unique ability to reset the Locality 4 PCR. It can also use HASH_DATA to send data to a TPM to be hashed and extended to the Locality 4 PCR. There is no header or other information that accompanies the data, for the FIFO interface. For the CRB interface, the first two bytes of the data contain the size of the data to be hashed. Upon receipt of HASH_END, a TPM will initialize the PCR, complete the hash, and extend the resultant value into the Locality 4 PCR, as defined in a *TPM 2.0 Library Specification*.

The Locality 4 PCR (PCR[17]) contains the first measurement of the Dynamic RTM for the Dynamic OS. Because the security of the Dynamic Launch is dependent solely on the reset and initial measurement in the Locality 4 PCR, access to Locality 4's extend operations should not have security implications.

It is expected that any PC Client platform is designed such that the platform protects Locality 4 access to a TPM, ensuring access only from platform components operating at Locality 4.

Note: For a D-RTM sequence (HASH_START/_DATA/_END) to occur, a TPM must have received a TPM2_Startup command prior to the HASH_START. If a TPM receives HASH_START after a _TPM_INIT but before a startup command, a TPM treats this as a S-HCRTM sequence.

The data written to a TPM_HASH_START and TPM_HASH_END interface registers of the FIFO interface has no significance and may be any value.

Note: For the CRB interface, an optimization allows both a TPM_HASH_DATA and TPM_HASH_END fields of a TPM_LOC_CTRL_4 register to be SET in the same write cycle.

This specification defines a persistent indicator in a TPM that allows a caller to detect that a Dynamic OS has been invoked regardless of whether the Dynamic OS is currently controlling the platform. This indication is done using the Establishment bit. The state of this bit upon any TPM2_Startup is 1 until the first DRTM sequence. The first DRTM sequence (which begins the chain of trust for the Dynamic OS) is signaled using HASH_START, which sets the Establishment bit to 0. See Sections 6.5.2.4 and 6.5.3.2.2.

DRAFT

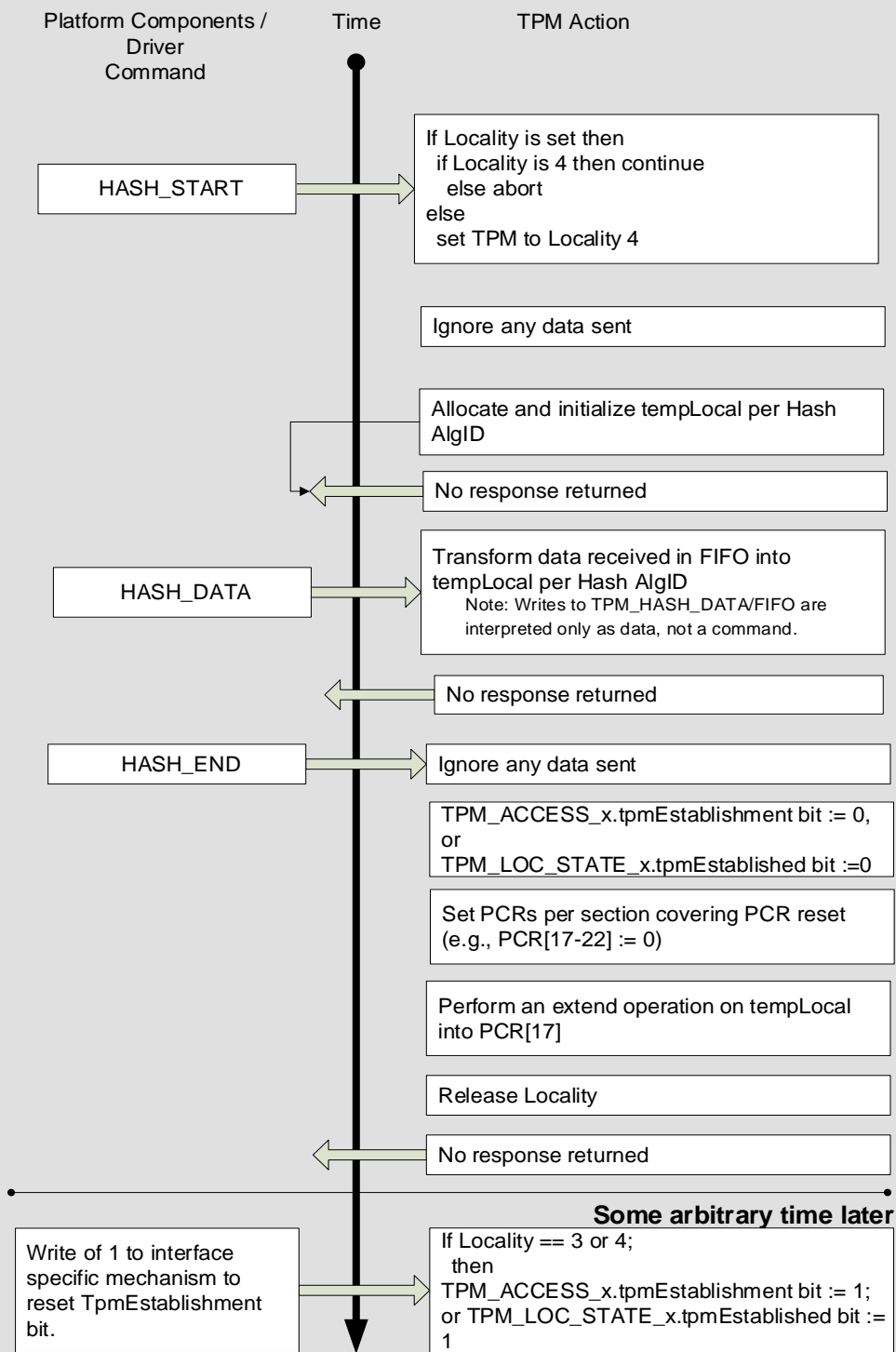


Figure 1 — Overview of D-RTM Measurement Sequence

End of informative comment

1. A D-RTM sequence is started by a HASH_START which occurs following a TPM2_Startup command.
 - a. A HASH_START is either:
 - i. In the FIFO interface, a successful write to a TPM_HASH_START interface register, or
 - ii. In the CRB interface, a successful write to a TPM_LOC_CTRL_4 interface register with a TPM_HASH_START field SET.
2. D-RTM data is provided by a HASH_DATA which occurs following a HASH_START
 - a. A HASH_DATA is either:
 - i. In the FIFO interface, a successful write to a TPM_HASH_DATA interface register, or
 - ii. In the CRB interface, a successful write of two or more bytes to the command buffer followed by a write to a TPM_LOC_CTRL_4 interface register with a TPM_HASH_DATA field SET.

Note: The first two bytes in the command buffer, in big endian notation, indicate the number of bytes to be hashed when HASH_DATA is received.

3. A D-RTM sequence is completed by a HASH_END that follows a HASH_START
 - a. A HASH_END is either:
 - i. In the FIFO interface, a successful write to a TPM_HASH_END interface register, or
 - ii. In the CRB interface, a successful write to a TPM_LOC_CTRL_4 interface register with a TPM_HASH_END field SET.
4. Upon receipt of HASH_START, a TPM SHALL follow the protocol below and perform the operations in the following pseudo-code to affect the resettable PCRs:

Note: While the resulting functionality presented by the steps below is normative, the actual operations and their sequence as presented here are informative. There is no requirement to perform the following operations exactly as shown. However implemented, the results are required to be the same as if a TPM were implemented as described below.

- a. HASH_START: Upon receipt of this interface command, a TPM SHALL:
 - i. If no Locality field is set, SET the active Locality field to indicate Locality 4.
 - ii. If a locality is active, and if the active Locality field is not 4, ignore this command.
 - iii. A TPM MAY CLEAR the Establishment bit.
 - iv. Clear the write buffer (FIFO only).

Note: If the FIFO is cleared as a result of relinquishing locality, this step may be omitted.

- v. If there is an exclusive session, a TPM SHALL have no exclusive session following the HASH_START.
- vi. Ignore any data component of this interface command.
- vii. Perform operations per *TPM 2.0 Library Specification* for the `_TPM_Hash_Start` indication.
- b. HASH_DATA: Upon receipt of this interface command, a TPM SHALL perform operations as defined in *TPM 2.0 Library Specification* for the `_TPM_Hash_Data` indication.
- c. HASH_END: Upon receipt of this interface command, a TPM SHALL:
 - i. CLEAR the Establishment bit if the Establishment bit was not CLEARED by HASH_START.

Note: See description of Bit Field: Establishment bit in Section 6.5.2.4 Access Register and Section 6.5.3.2.1 Locality State Register.

- ii. Set to 0 all PCR which are reset by a D-RTM event as indicated by Table 10.
- iii. Perform the hash functions as if TPM2_PCR_Event command was being used.

Note: This is consistent with the Hash Interface Command in TPM 1.2.

- iv. Perform operations as defined in *TPM 2.0 Library Specification* for the `_TPM_Hash_End` indication.

- v. Clear active Locality.

Note: The write buffer is cleared as a result of relinquishing locality for both the FIFO and CRB interfaces.

- d. After successful completion of HASH_START and before HASH_END:
 - i. For the FIFO interface, all cycles and commands other than writes to a TPM_HASH_DATA and TPM_HASH_END interface registers SHALL be ignored until HASH_END
 - ii. For the CRB interface, all cycles and commands other than writes to the Command Buffer and TPM_LOC_CTRL_4 registers SHALL be ignored until HASH_END.
- e. Upon any error in the above steps a TPM SHALL release locality and enter Failure Mode.

Note: No response packet is returned for HASH_START, HASH_DATA, or HASH_END.

5.3.2 S-HCRTM Sequence Before TPM2_Startup and TPM2_Startup without S-HCRTM

Start of informative comment

The S-HCRTM is started in the earliest stage of platform boot and is initiated by a CPU. The S-HCRTM sequence only applies to PCR 0 if the sequence is initiated prior to a TPM2_Startup command. The actions of the sequence are identical to that of a D-RTM sequence except for the value to which the PCR is initialized and when the PCR is initialized and extended.

The error handling for the S-HCRTM sequence is identical to the error handling described in 5.3.1 D-RTM Execution Sequence.

End of informative comment

1. A TPM2_Startup command SHALL come from Locality 0 or 3, else a TPM SHALL return TPM_RC_Locality.
2. Upon receipt of a HASH_START, a TPM follows the protocol below and performs the operations in the following pseudo-code to affect PCR 0:

Note: While the resulting functionality presented by the steps below is normative, the actual operations and their sequence as presented here are informative. There is no requirement to perform the following operations exactly as shown. However implemented, the results are required to be the same as if a TPM were implemented as described below.

HASH_START: Upon receipt of this interface command, a TPM SHALL:

- i. If no locality field is set, set the active locality field to indicate Locality 4.
- ii. If a locality is active, and if the active locality field is not 4, ignore this command.
- iii. Perform operations as defined in TPM 2.0 Library Specification _TPM_Hash_Start indication.
- b. HASH_DATA: Upon receipt of this interface command, a TPM SHALL perform operations as defined in TPM 2.0 Library Specification _TPM_Hash_Data indication.

Note: For the CRB interface, bytes 0 and 1 of the data written to the command buffer contain the length of the subsequent data to be extended.

- c. HASH_END: Upon receipt of this interface command, a TPM SHALL:
 - i. Perform operations as defined in TPM 2.0 Library Specification _TPM_Hash_End
 - ii. Clear active Locality.

5.3.3 Timing and Protocol

Start of informative comment

The D-RTM and S-HCRTM sequences execute within a resource-restricted environment which is among the reasons the HASH_START/_DATA/_END protocol is used rather than the more obvious TPM command ordinals (e.g., TPM2_PCR_Extend). It is also difficult and unnecessary for this environment to use the register-based protocols. Therefore, during the Locality 4 HASH_START/_DATA/_END sequence, the only method to throttle commands to a

TPM uses the bus wait mechanism. (There is no data returning from TPM within this environment.) Specifically, a TPM uses the LPC bus “long wait” sync (using LPC terms) or SPI wait cycles as defined in Section 7.4.5 Flow Control to indicate to the “host” that it is unable to accept more data.

This environment also may not be conducive to “timeouts” and may be very susceptible to delays or hangs. It is important that a TPM be designed to avoid excessive delays and should not cause the bus to hang during this time.

When a TPM has several hash algorithms and multiple active PCR banks, a TPM takes longer to finalize the Hash sequence.

Prior to PTP 1.06, the time limit for responding to HASH_DATA in Normative 3 was 250 microseconds.

Note: For CRB based TPMs, the D-RTM and S-HCRTM sequences are controlled via a TPM_CRB_LOC_CTRL_4 register, which has no flow control. For integrated or memory based TPMs, caution is advised when sending multiple, large _HASH_DATA commands, as there is no way to know whether a TPM received all the portions.

End of informative comment

1. During the _HASH_START, _HASH_DATA, and _HASH_END sequence, a TPM SHALL use the appropriate bus wait mechanism (LPC bus “long wait sync” or SPI “wait cycle”) to indicate its inability to accept more commands or data. While a TPM MAY set a TPM_STS_x or TPM_CRB_CTRL_REQ_x fields they are “undefined” during these commands (i.e., will likely not be read and will not be honored).
2. A TPM SHALL comply with the command duration and Timeout requirements defined in Section 6.5.1.3 Command Duration and Section 6.5.1.4 Interface Timeouts.
3. A TPM SHALL accept a minimum data size of 1kB for a _HASH_DATA interface sequence.

6 TPM Software Interface

6.1 Interface Type

Start of informative comment

This specification defines a new Software interface to a TPM for TPM 2.0, in addition to the FIFO interface. This interface, the Command Response Buffer Interface, has been defined so that it may be implemented in a TPM which also contains a FIFO interface. The CRB Interface is intended to be physical-bus agnostic, so that it could be implemented on an LPC or SPI interface, as specified in this specification or on another physical interface not specified. For a TPM to be compliant with this specification, however, it is required to implement at least one of the interfaces defined by this specification.

The physical register spaces for both FIFO and CRB are specified in Section 6.3 TPM Register Space. Register space with functions common to both interfaces is specified in Section 6.5.1 Interface Agnostic Functions. The behavior of the CRB Interface is specified in Section 6.5.3 CRB Interface Requirements. In the subsequent sections, functionality that is interface-independent precedes the interface-specific functionality. Where a function is common to both interfaces, but there are interface-specific requirements, the requirements are documented in the interface-specific section. For example, the concepts of locality are common to both interfaces, but the mechanisms to invoke locality are interface specific.

End of informative comment

1. A TPM compliant with this specification SHALL implement at least one of the following interfaces:
 - a. Command Response Buffer (CRB) Interface, or
 - b. FIFO Interface
2. A TPM SHALL implement the InterfaceType field in the interface specific Interface Identifier register, TPM_INTERFACE_ID_x for FIFO and TPM_CRB_INTF_ID_x for CRB.
3. A TPM which supports both interface types SHALL expose only one Interface at a time.
4. The mechanism for switching between interfaces SHALL be implemented as defined in Section 6.4.2 Interface Identifier Register.

6.2 Locality

Start of informative comment

Locality Priority is described in Section 3.2 [Locality](#).

End of informative comment

6.2.1 TPM Locality Levels

Start of informative comment

TPM 2.0 supports five levels of locality: Locality None and Localities 0-4.

The usage of PCRs with respect to locality is defined in Section 4.6.1 PCR Attributes.

For the platform, the locality level is indicated by the address used along with a TPM bus Start cycle. For system software on an X86 architecture, a TPM has a 64-bit address of 0x0000_0000_FED4_xxxx. On non-X86 architectures, e.g., ARM, a TPM may have a different base than 0x0000_0000_FED4_xxxx. In this case, the upper octets may vary, but the rest are required to be the same for compatibility on the physical interface. Note, unless there is a compelling reason to modify a TPM base address, platform implementers should map a TPM to the 0x0000_0000_FED4_xxxx base address. The mapped base address is populated in a TPM Device Object in the ACPI table, as defined in the TCG ACPI Specification Family for TPM 1.2 and 2.0. On LPC, the chipset passes the least significant 16 bits to a TPM. On SPI, the chipset passes the least significant 24 bits to a TPM. The upper bytes will be used by the chipset to select a TPM's SPI CS# signal. Table 13 shows the locality based on the 16 least significant address bits and assume that either the LPC TPM sync or SPI TPM CS# is used. TPM commands, e.g., TPM2_PCR_Reset, may also require locality. A TPM, upon receipt of each command, sets (based on a TPM register Address) the locality for the command. Locality, as presently defined, may be Locality 0 through 4.

End of informative comment**Table 13 — Locality Address Definitions**

System/Software Address (Informative)	TPM Address on LPC (Using TPM START Cycle)	TPM Address on SPI	Locality
FED4_0xxxh	0xxxh	D4_0xxxh	0
FED4_1xxxh	1xxxh	D4_1xxxh	1
FED4_2xxxh	2xxxh	D4_2xxxh	2
FED4_3xxxh	3xxxh	D4_3xxxh	3
FED4_4xxxh	4xxxh	D4_4xxxh	4

NOTE: This specification uses the following convention for describing addresses without distinguishing between the physical interface address. System/software addresses in this specification are generally referred to as `xxD4_xxxxh`. Unless otherwise noted, this system/software address should be interpreted as = `xxxxh` on an LPC interface, or `D4_xxxxh` on an SPI interface.

Start of informative comment**Note on locality priority:**

The statements in Section 3.2 regarding locality hierarchy notwithstanding, the selection of localities has a priority. If two localities have requested use of a TPM when the current locality relinquishes it, the locality with the highest priority gets access to a TPM. The locality's priority increases as its locality number increases. i.e., Locality 0 has the lowest priority while Locality 4 has the highest.

Note on Locality 4 management:

Locality 4 accesses are controlled by trusted hardware responsible for maintaining the D-RTM, and Software should not use Locality 4 commands. Trusted hardware should be implemented so that Locality 4 operations are not accessible to Software.

If the active locality is a locality other than 4, `HASH_START` is ignored. If there is no locality set, `HASH_START` makes Locality 4 the active locality. Once the `HASH_DATA` sequence is completed at Locality 4, `HASH_END` releases Locality 4 and returns a TPM to a "free" state.

End of informative comment

1. A TPM SHALL maintain the relationship between locality and the locality attribute as defined in Table 14.
2. The Locality with the highest numeric value has the highest priority, when assigning locality in response to a host Software request to use a TPM using either the locality request method or Seize method, i.e., Locality 1 has a higher priority than Locality 0.
3. When a TPM is at the "free" state, it SHALL accept a write to any of a `TPM_HASH_x` registers (FIFO interface) or a `TPM_LOC_CTRL_4` register (CRB interface).

Table 14 — Relationship between Locality and Locality Attribute

Locality	Value of Locality Indicator
Locality 0	01h
Locality 1	02h
Locality 2	04h
Locality 3	08h
Locality 4	10h

6.2.2 Locality Uses

Start of informative comment

Usage of Locality 0 PCRs is determined by the *TCG PC Client Specific Implementation Specification*. Usage of Locality 1-3 PCRs is reserved for uses which are outside the purview of this specification.

The idea behind locality is that certain combinations of Software and hardware are allowed more privileges than other combinations. For instance, the highest level of locality might be cycles that only hardware could create.

While higher localities may exist, Locality 4 is the highest locality level defined. These cycles are generated by hardware in support of the D-RTM. Cycles which require Locality 4 would include things such as the HASH_START/_DATA/_END interface commands.

As an example, assume a platform, including Software, has an operating system based on the Static RTM or Static OS, based on either using no PCRs or the set of non-resettable PCRs (0-15), and trusted Software, the dynamically launched operating system, or Dynamic OS, which uses the resettable PCRs (17-22). In this case, there is a need to differentiate cycles originating from the two operating systems. Localities 1-3 are used by the Dynamic OS for its transactions. The Dynamic OS has created certain values in the resettable PCRs. Only the Dynamic OS should be able to issue commands based on those PCRs. The Static OS uses Locality 0.

The non-resettable PCRs (i.e., PCR[0-15]) are not part of the Dynamic OS's domain, so Locality 0 transactions can freely use those PCRs, but must be prevented from resetting or extending PCRs used by the Dynamic OS (see Section 4.6.1 PCR Attributes for the PCR's which are resettable by the Dynamic OS).

Note to Platform and OS Implementers:

Each RTM (e.g., the D-RTM) has a root PCR associated with it. The fundamental trusted boot requirement is that when the RTM is initiated/reset its associated root PCR must also be reset. Conversely, the root PCR must never be reset unless its associated RTM is also initialized/reset. When launching the Dynamic OS, the root dynamic PCR must only be reset when the D-RTM is initiated/reset.

A TPM architecture doesn't provide a TPM with a method for controlling access to any of its localities, therefore, controlling access to the various localities is up to either the platform components or the OS. However, even the Dynamic OS must not be allowed to reset the root Dynamic PCR because the Dynamic OS is what is measured into this PCR. The platform components must restrict access to Locality 4 to only the D-CRTM components. This is to protect the resetting of the root Dynamic PCR (i.e., PCR[17]). (Note that because some TPMs have implemented the command FIFO for Locality 4, the platform must protect the entire Locality 4 address range to prevent unauthorized Software from executing a TPM2_PCR_Reset command on PCR[17] at Locality 4.)

While the Dynamic OS is executing, the platform components may provide Software access to localities other than 4. In this case, if the Dynamic OS requires protection for these localities, it must protect them using methods such as virtual memory management (i.e., paging).

End of informative comment

1. A TPM SHALL enforce locality access to each TPM resource that requires locality (such as PCRs) or uses TPM2_PolicyLocality in its authorization policy.

6.3 TPM Register Space

6.3.1 TPM Register Space Decode

Start of informative comment

Many of the registers in a TPM Register Space are defined using a contiguous address range within a given locality. There are common requirements for decoding the address space between the FIFO register space and the CRB register space. These common requirements are documented in this section. The actual address space for FIFO and CRB is different and there are some unique restrictions. The interface specific information is documented in the subsequent sections for each interface.

FIFO Register Space Decode

Most of the FIFO registers are accessed with a TPM decoding all addresses within the specified address ranges. For registers with the same function for different localities, the address range for one locality is not contiguous with the address range for a different locality. Some of these registers that are defined separately with separate address ranges, e.g., a TPM_STS_x register, may be mirrored such that each separate address range (for example 0x001A to 0x0018 for TPM_STS_0 and 0x101A_0x1018 for TPM_STS_1) points to the same physical register.

Another one of the registers which is defined as having five addresses is a TPM data register (TPM_DATA_FIFO_x). For this register, the addresses within this range may be aliased to one internal register.

Note that the addresses allocated for the Locality 4 HASH* commands do not exist in Localities 0-3.

CRB Register Space Decode

The CRB address space is a contiguous space that may be instantiated in hardware or in ACPI AddressRangeReserved memory. When a TPM implements a CRB in hardware, the register space is defined in the Section 6.5.3 CRB Interface Requirements.

General Address Space Decode Considerations

The LPC bus transfers a single byte per transaction to any of the registers. The chipset may, for performance reasons, send 4 bytes for each transaction. This will appear as four distinct LPC bus transactions, with each incrementing the address by 1 byte with each set of transactions starting with the least significant byte – thus being little-endian. Because a TPM can accept only 1 byte per transaction, a TPM must ignore the 2 least significant bits of the address for the data registers thus receiving the data serially. However, a TPM should not require or expect that each address is incremented modulo-4.

Unlike LPC, the SPI bus does not limit transfers to a single byte. The extended size data register (TPM_XDATA_FIFO_x) and CRB data buffer allow a single write to offset 0x0080h up to the maximum transfer size reported by TPM_INTF_Capability_x.DataTransferSizeSupport, without requiring Software to increment the address. I2C has no such transfer size restriction, so the extended size data register is not required. It is technically possible to send a single transaction on SPI that spans more than one register in a TPM's address space. Software should never attempt to access multiple registers in a single transaction. Software should access each register in unique, individual transactions and not attempt to cross register boundaries. Software may access only part of a register, e.g. read or write one byte of a 4-byte register. Software should not initiate a transaction that is either larger than the register size (2-byte access to a 1-byte register), or that extends beyond the defined register boundary (2-byte transaction to offset 0x3 of a register defined as existing within the address range of 0x0 to 0x3). This is simply good Software behavior, and these guidelines are not specific to TPM transactions but apply to all hardware and Software.

Because a TPM must be designed to handle cases where software behaves badly to avoid leaking TPM protected information, this specification defines behavior for a TPM if a transaction crosses multiple registers. TPM vendors should design their hardware so that bad Software does not impact the state or security of a TPM. Specific error behavior is specified in Section 6.5.1.8 Errors.

End of informative comment

1. A TPM compliant with this specification SHALL implement an Interface Identifier register:
 - a. A TPM which supports only the FIFO interface SHALL support a TPM_INTERFACE_ID_x register.

- b. A TPM which supports a CRB interface SHALL support a TPM_CRB_INTF_ID_x register.
- 2. For SPI, if a TPM receives an access request with a length that exceeds the size of the register specified in the transaction address:
 - a. Reads:
 - i. A TPM SHALL return the data for the register designated by the start address.
 - ii. A TPM MAY return the data for additional, adjacent registers within the targeted address range, or a TPM MAY return dummy data for additional, adjacent registers within the targeted address range.
 - b. Writes:
 - i. A TPM SHALL update the register designated by the start address.
 - ii. A TPM MAY update additional, adjacent registers within the targeted address range.
 - iii. A TPM SHALL NOT change the state of adjacent registers if the writes to that register are dropped.
 - c. A TPM SHOULD NOT abort (as defined in Section 6.5.1.1 Bus Aborts) an entire transaction that crosses a register boundary.
 - d. When a transaction crosses a register boundary, a TPM SHALL NOT allow data from that transaction to corrupt future SPI transactions.
- 3. For the FIFO data buffers:
 - a. TPM_DATA_FIFO_x
A TPM SHALL ignore the 2 least significant bits of the address for these registers and accept each byte received by any of the addresses within this register as a single transfer to the base address.
 - b. TPM_XDATA_FIFO_x
A TPM SHALL accept transactions to offset 0x0080h that are of any length from 1 byte to the maximum supported length (as reported in the Interface Capability register, Section 6.5.2.7 Interface Capability).
- 4. A TPM MAY accept transactions to offset 0x0080h that are of lengths larger than the maximum supported length (as reported in the Interface Capability register, see Section 6.5.2.7 Interface Capability), based on the existing interface protocol using TPM_STS_x.burstCount and TPM_STS_x.Expect as defined in Section 6.5.2.4 Access Register.
 - a. All other registers:
 - i. A TPM SHALL fully decode the address down to the byte level (unless otherwise specified).
 - ii. A TPM SHALL interpret registers that have multiple addresses as follows:
 - 1. The lowest address within the set of addresses contains the least significant byte with the bits incrementing to each successive address up to the highest address which contains the most significant byte.

Note: Addresses are implemented as described in Table 15..

- 5. For the CRB Interface, as all registers are aligned on either 32-bit or 64-bit address boundaries, the following restrictions apply:
 - a. For an access with a start address for an address aligned on a 32-bit boundary with a length larger than 32 bits:
 - i. For write transactions,
 - ii. A TPM MAY update the register with the start address equal to the transaction start address and wholly contained within the transaction address.
 - 1. A TPM MAY ABORT (as defined in Section 6.5.1.1 Bus Aborts) the transaction.
 - 2. A TPM SHALL not update any register whose start address is not equal to the start address of the transaction.
 - iii. For read transactions,

1. A TPM MAY return the response data for the register with the start address equal to the transaction start address and wholly contained within the transaction address.
2. A TPM MAY ABORT (as defined in Section 6.5.1.1 Bus Aborts) the transaction.

Start of informative comment

Table 15 — Example Bit-to-Address Mapping

Address	Data Bit Position	Example
0x00000008	7:0	0000 0000 0000 0000 0000 0000 1000
0x00000900	15:8	0000 0000 0000 0000 0000 1001 0000 0000
0x000A0000	23:16	0000 0000 0000 1010 0000 0000 0000 0000
0x0B000000	31:24	0000 1011 0000 0000 0000 0000 0000 0000

End of informative comment

6.3.2 Register Space Addresses

Start of informative comment

Table 16 lists a comparison of the addresses decoded by a TPM when FIFO or CRB (in hardware) is implemented.

End of informative comment

Table 16 — Allocation of Register Space for FIFO and CRB Access

Offset	FIFO Register Name	CRB Register Name	
Locality 0			
0000h	TPM_ACCESS_0	TPM_LOC_STATE_0	
0001h	Reserved		
0002h			
0003h			
0007h-0004h			Reserved
000Bh-0008h	TPM_INT_ENABLE_0	TPM_LOC_CTRL_0	
000Ch	TPM_INT_VECTOR_0	TPM_LOC_STS_0	
000Fh-000Dh	Reserved	TPM_DATA_CSUM_ENABLE_0	
0013h-0010h	TPM_INT_STATUS_0		
0017h-0014h	TPM_INTF_CAPABILITY_0		
001Bh-0018h	TPM_STS_0		
0023h-001Ch	Reserved		
0027h-0024h	TPM_DATA_FIFO_0	Reserved	
002Fh-0028h	Reserved		
0033h-0030h	TPM_INTERFACE_ID_0		
0037h-0034h	TPM_DATA_CSUM_ENABLE_0		
003Bh-0038h	TPM_DATA_CSUM_0		
003Fh-003Ch	Reserved	TPM_CRB_INTF_ID_0	
0043h-0040h	Reserved		TPM_CRB_CTRL_REQ_0
0047h-0044h			TPM_CRB_CTRL_STS_0
004Bh-0048h			TPM_CRB_CTRL_CANCEL_0
004Fh-004Ch			TPM_CRB_CTRL_START_0
0053h-0050h		TPM_CRB_INT_ENABLE_0	
0057h-0054h		TPM_CRB_INT_STS_0	
005Bh-0058h		TPM_CRB_CTRL_CMD_SIZE_0	
005Fh-005Ch		TPM_CRB_CTRL_CMD_LADDR_0	
0063h-0060h		TPM_CRB_CTRL_CMD_HADDR_0	
0067h-0064h		TPM_CRB_CTRL_RSP_SIZE_0	
006Fh-0068h		TPM_CRB_CTRL_RSP_ADDR_0	
007Fh-0070h		Reserved	
0083h-0080h		TPM_XDATA_FIFO_0	TPM_CRB_DATA_BUFFER_0
0880h-0084h		Reserved	
0EFFh-0881h			
0F03h-0F00h	TPM_DID_VID_0	Reserved	
0F04h	TPM_RID_0		
0FFFh-0F90h	Reserved		

Offset	FIFO Register Name	CRB Register Name	
Locality 1			
1000h	TPM_ACCESS_1	TPM_LOC_STATE_1	
1001h	Reserved		
1002h			
1003h			
1007h-1004h		Reserved	
100Bh-1008h	TPM_INT_ENABLE_1	TPM_LOC_CTRL_1	
100Ch	TPM_INT_VECTOR_1	TPM_LOC_STS_1	
100Fh-100Dh	Reserved		
1013h-1010h	TPM_INT_STATUS_1	TPM_DATA_CSUM_ENABLE_1	
1017h-1014h	TPM_INTF_CAPABILITY_1	TPM_DATA_CSUM_1	
101Bh-1018h	TPM_STS_1	Reserved	
1023h-101Ch	Reserved		
1027h-1024h	TPM_DATA_FIFO_1		
102Fh-1028h	Reserved		
1033h-1030h	TPM_INTERFACE_ID_1	TPM_CRB_INTF_ID_1	
1037h-1034h	TPM_DATA_CSUM_ENABLE_1		
103Bh-1038h	TPM_DATA_CSUM_1	Reserved	
103Fh-103Ch	Reserved		
1043h-1040h	Reserved	TPM_CRB_CTRL_REQ_1	
1047h-1044h		TPM_CRB_CTRL_STS_1	
104Bh-1048h		TPM_CRB_CTRL_CANCEL_1	
104Fh-104Ch		TPM_CRB_CTRL_START_1	
1053h-1050h		TPM_CRB_INT_ENABLE_1	
1057h-1054h		TPM_CRB_INT_STS_1	
105Bh-1058h		TPM_CRB_CTRL_CMD_SIZE_1	
105Fh-105Ch		TPM_CRB_CTRL_CMD_LADDR_1	
1063h-1060h		TPM_CRB_CTRL_CMD_HADDR_1	
1067h-1064h		TPM_CRB_CTRL_RSP_SIZE_1	
106Fh-1068h		TPM_CRB_CTRL_RSP_ADDR_1	
107Fh-1070h		Reserved	
1083h-1080h		TPM_XDATA_FIFO_1	TPM_CRB_DATA_BUFFER_1
1880h-1084h		Reserved	
1EFFh-1881h			
1F03h-1F00h		TPM_DID_VID_1	Reserved
1F04h		TPM_RID_1	
1FFFh-1F05h	Reserved		

Offset	FIFO Register Name	CRB Register Name	
Locality 2			
2000h	TPM_ACCESS_2	TPM_LOC_STATE_2	
2001h	Reserved		
2002h			
2003h			
2007h-2004h			Reserved
200Bh-2008h	TPM_INT_ENABLE_2	TPM_LOC_CTRL_2	
200Ch	TPM_INT_VECTOR_2	TPM_LOC_STS_2	
200Fh-200Dh	Reserved		
2013h-2010h	TPM_INT_STATUS_2	TPM_DATA_CSUM_ENABLE_2	
2017h-2014h	TPM_INTF_CAPABILITY_2	TPM_DATA_CSUM_2	
201Bh-2018h	TPM_STS_2	Reserved	
2023h-201Ch	Reserved		
2027h-2024h	TPM_DATA_FIFO_2		
202Fh-2028h	Reserved		
2033h-2030h	TPM_INTERFACE_ID_2	TPM_CRB_INTF_ID_2	
2037h-2034h	TPM_DATA_CSUM_ENABLE_2		
203Bh-2038h	TPM_DATA_CSUM_2	Reserved	
203Fh-203Ch	Reserved		
2043h-2040h	Reserved	TPM_CRB_CTRL_REQ_2	
2047h-2044h		TPM_CRB_CTRL_STS_2	
204Bh-2048h		TPM_CRB_CTRL_CANCEL_2	
204Fh-204Ch		TPM_CRB_CTRL_START_2	
2053h-2050h		TPM_CRB_INT_ENABLE_2	
2057h-2054h		TPM_CRB_INT_STS_2	
205Bh-2058h		TPM_CRB_CTRL_CMD_SIZE_2	
205Fh-205Ch		TPM_CRB_CTRL_CMD_LADDR_2	
2063h-2060h		TPM_CRB_CTRL_CMD_HADDR_2	
2067h-2064h		TPM_CRB_CTRL_RSP_SIZE_2	
206Fh-2068h		TPM_CRB_CTRL_RSP_ADDR_2	
207Fh-2070h		Reserved	
2083h-2080h		TPM_XDATA_FIFO_2	TPM_CRB_DATA_BUFFER_2
2880h-2084h		Reserved	
2EFFh-2881h			
2F03h-2F00h		TPM_DID_VID_2	Reserved
2F04h		TPM_RID_2	
2FFFh-2F05h	Reserved		

Offset	FIFO Register Name	CRB Register Name	
Locality 3			
3000h	TPM_ACCESS_3	TPM_LOC_STATE_3	
3001h	Reserved		
3002h			
3003h			
3007h-3004h			Reserved
300Bh-3008h	TPM_INT_ENABLE_3	TPM_LOC_CTRL_3	
300Ch	TPM_INT_VECTOR_3	TPM_LOC_STS_3	
300Fh-300Dh	Reserved	Reserved	
3013h-3010h	TPM_INT_STATUS_3		
3017h-3014h	TPM_INTF_CAPABILITY_3		
301Bh-3018h	TPM_STS_3		
3023h-301Ch	Reserved		
3027h-3024h	TPM_DATA_FIFO_3	TPM_CRB_INTF_ID_3	
302Fh-3028h	Reserved		
3033h-3030h	TPM_INTERFACE_ID_3		
3037h-3034h	TPM_DATA_CSUM_ENABLE_3		
303Bh-3038h	TPM_DATA_CSUM_3		
303Fh-303Ch	Reserved	Reserved	
3043h-3040h	Reserved	TPM_CRB_CTRL_REQ_3	
3047h-3044h		TPM_CRB_CTRL_STS_3	
304Bh-3048h		TPM_CRB_CTRL_CANCEL_3	
304Fh-304Ch		TPM_CRB_CTRL_START_3	
3053h-3050h		TPM_CRB_INT_ENABLE_3	
3057h-3054h		TPM_CRB_INT_STS_3	
305Bh-3058h		TPM_CRB_CTRL_CMD_SIZE_3	
305Fh-305Ch		TPM_CRB_CTRL_CMD_LADDR_3	
3063h-3060h		TPM_CRB_CTRL_CMD_HADDR_3	
3067h-3064h		TPM_CRB_CTRL_RSP_SIZE_3	
306Fh-3068h		TPM_CRB_CTRL_RSP_ADDR_3	
307Fh-3070h		Reserved	TPM_CRB_DATA_BUFFER_3
3083h-3080h		TPM_XDATA_FIFO_3	
3880h-3084h		Reserved	
3EFFh-3881h		Reserved	Reserved
3F03h-3F00h	TPM_DID_VID_3		
3F04h	TPM_RID_3		
3FFFh-3F05h	Reserved		

Offset	FIFO Register Name	CRB Register Name	
Locality 4			
4000h	TPM_ACCESS_4	TPM_LOC_STATE_4	
4001h	Reserved		
4002h			
4003h			
4007h-4004h		Reserved	
400Bh-4008h	TPM_INT_ENABLE_4	TPM_LOC_CTRL_4	
400Ch	TPM_INT_VECTOR_4	TPM_LOC_STS_4	
400Fh-400Dh	Reserved		
4013h-4010h	TPM_INT_STATUS_4	TPM_DATA_CSUM_ENABLE_4	
4017h-4014h	TPM_INTF_CAPABILITY_4	TPM_DATA_CSUM_4	
401Bh-4018h	TPM_STS_4	Reserved	
401Fh-401Ch	Reserved		
4023h-4020h	TPM_HASH_END		
4027h-4024h	TPM_HASH_DATA / TPM_DATA_FIFO_4		
402Fh-4028h	TPM_HASH_START		
4033h-4030h	TPM_INTERFACE_ID_4	TPM_CRB_INTF_ID_4	
4037h-4034h	TPM_DATA_CSUM_ENABLE_4		
403Bh-4038h	TPM_DATA_CSUM_4	Reserved	
403Fh-403Ch	Reserved		
4043h-4040h	Reserved	TPM_CRB_CTRL_REQ_4	
4047h-4044h		TPM_CRB_CTRL_STS_4	
404Bh-4048h		TPM_CRB_CTRL_CANCEL_4	
404Fh-404Ch		TPM_CRB_CTRL_START_4	
4053h-4050h		TPM_CRB_INT_ENABLE_4	
4057h-4054h		TPM_CRB_INT_STS_4	
405Bh-4058h		TPM_CRB_CTRL_CMD_SIZE_4	
405Fh-405Ch		TPM_CRB_CTRL_CMD_LADDR_4	
4063h-4060h		TPM_CRB_CTRL_CMD_HADDR_4	
4067h-4064h		TPM_CRB_CTRL_RSP_SIZE_4	
406Fh-4068h		TPM_CRB_CTRL_RSP_ADDR_4	
407Fh-4070h			Reserved
4083h-4080h		TPM_XDATA_FIFO_4	TPM_CRB_DATA_BUFFER_4
4880h-4084h		Reserved	
4EFFh-4881h			
4F03h-4F00h		TPM_DID_VID_4	Reserved
4F04h	TPM_RID_4		
4FFh-4F05h	Reserved		
Non-Locality Specific Registers			
5FFFh-5000h	Reserved	Reserved	

Subsequent sections provide implementation details on the defined registers for both FIFO and CRB interfaces. See Sections 6.5.2 FIFO Interface Requirements and 6.5.3 CRB Interface Requirements.

6.4 System Interaction and Flows

6.4.1 FIFO Configuration Registers

6.4.1.1 DID/VID Register

Table 17 — DID/VID Register

Abbreviation:		TPM_DID_VID_x	
General Description:		Vendor and Device ID for a TPM	
Default		Vendor specific	
Bit Descriptions:			
31:16	Read Only	DID	Device ID – vendor specific
15:0	Read Only	VID	Vendor ID – Assigned by TCG Administration. This is represented within the register in big-endian format. For example, a vendor ID of 0x1234 would be represented as: Bits 7:0 = 34 (0011 0100); Bits 15:8 = 12 (0001 0010).

6.4.1.2 RID Register

Table 18 — RID Register

Abbreviation:		TPM_RID_x	
General Description:		Revision ID for a TPM	
Default		Specific to each revision	
Bit Descriptions:			
7:0	Read Only	RID	Revision ID – specifies the revision of the component

6.4.2 Interface Identifier Register

Start of informative comment

The Interface Identifier register is defined for both the legacy FIFO interface and the new CRB interface to allow TPM vendors to support both interfaces, but not all functions are present in the FIFO that are present in the CRB version. The CRB Interface Identifier includes DID/VID/RID registers, which are in a separate address space in the FIFO interface. Software can query this register to determine which interface a TPM supports, the Interface Version and what Interface type is currently enabled. The Interface Identifier registers are aliased across localities.

TPMs implemented to support PC Client TPM Interface Specification 1.3 [15] or earlier will return 1111b for InterfaceType. If this field returns 1111b, a TPM may be a TPM family 2.0 implemented with a FIFO interface compliant with TIS 1.3. If this is the case, PTP specific features will be implemented as part of a TPM_INTF_CAPABILITY_x register, not this register.

End of informative comment

6.4.2.1 FIFO Interface Identifier Register

Table 19 — FIFO Interface Identifier Register

Abbreviation:		TPM_INTERFACE_ID_x	
General Description:		Interface Identifier Register	
Default		Specific to each revision	
Bit Descriptions:			
31:24	Read Only	Reserved	Reserved
23	Read Only	CapSPICSUM	0 – A TPM does not support calculation of a checksum 1 – A TPM supports calculation of a checksum for command and response. The capability is enabled via a TPM_DATA_CSUM_ENABLE register. See Section 6.5.1.8.1 TPM_DATA_CSUM_ENABLE
22:20	Read Only	Reserved	Reads return 0
19	Read Write	IntfSelLock	0 – A write of this value is ignored 1 – A write of this value locks the InterfaceSelector field and prevents further changes. Field is reset to 0 on _TPM_INIT
18:17	Read Write	InterfaceSelector	00 – A write of this value changes the selected interface to TIS 01 – A write of this value changes the selected interface to CRB Writes to this field take effect on next _TPM_INIT. Other values are reserved. This field may be written only if IntfSelLock is 0.
16:15	Read Only	CapIFRes	Reserved for future interfaces (since InterfaceSelector is a two-bit field), reads return 0
14	Read Only	CapCRB	0 – CRB interface is not supported. 1 – CRB interface is supported and may be selected.
13	Read Only	CapTIS	0 – TIS interface is not supported. 1 – TIS interface is supported and may be selected.
12:9	Read Only	Reserved	Reserved Reads return 0
8	Read Only	CapLocality	0 – This interface supports Locality 0 only. 1 – This interface supports 5 localities (Localities 0-4).
7:4	Read Only	InterfaceVersion	0010 – CRB interface version 1 with support for TPM Library 1.59. See Table 21 — CRB Historical Interface Versions. 0000 – FIFO interface for TPM2.0.
3:0	Read Only	InterfaceType	0000 – FIFO interface as defined in PTP for TPM 2.0 is active. 0001 – CRB interface is active. 1111 – FIFO interface as defined in TIS1.3 is active (all other fields of this register are don't care).

Field: Interface Type

Start of informative comment

This field identifies the interface type currently active. While some TPMs may support either of these interfaces for product requirement and marketing purposes, only one interface can be active at a time. The selection of the interface is TPM Vendor Specific.

This field applies to the entire TPM address range within the scope of the Interface Type. For example, if the Interface Type is FIFO, the FIFO Interface spans all the localities defined for the FIFO Interface, which is addresses `xxD4_0000h – xxD4_4FFFh`.

The state of this field governs the behavior of the rest of this register. If Interface Type indicates a TIS 1.3 style Interface, no other field in this register is valid. The capabilities of the interface as defined in `TPM_INTF_CAPABILITY_x` determine what is supported by the interface.

Note: A TPM supporting the PC Client TPM Interface Specification v1.2 [15] compliant FIFO interface will report supported capabilities in a `TPM_INTF_CAPABILITY_x` register.

Note: If this field is set to 1111b, a TPM may have implemented features from this specification on an interface built to comply with the TCG PC Client TPM Interface Specification version 1.3. If so, the features will be described in a `TPM_INTF_CAPABILITY_x` register, and this register will not be implemented.

End of informative comment

1. A value of 1111b in this field SHALL be interpreted to mean a TPM supports a PC Client TPM Interface Specification v1.3 [15] compliant FIFO interface.
2. If a TPM supports this specification, the value of this field SHALL NOT be 1111b.
3. Writes to this field SHALL be ignored.
4. If this field is set to 0000b:
 - a. A TPM SHALL correctly report all other capabilities for `TPM_INTERFACE_ID_x` fields.
 - b. A TPM SHALL support `TPM_INTERFACE_ID_x.InterfaceVersion`, which SHALL be 0h for the FIFO interface.
 - c. A TPM MAY support `TPM_INTF_CAPABILITY_x.InterfaceVersion`
5. If this field is set to 0001b:
 - a. A TPM SHALL correctly report all other capabilities for `TPM_INTERFACE_ID_x` fields.
 - b. A TPM SHALL support `TPM_INTERFACE_ID_x.InterfaceVersion`, which SHALL be 0010b for the CRB interface.
 - c. A TPM SHALL NOT support `TPM_INTF_CAPABILITY_x.InterfaceVersion`.
6. If this field is set to 1111b, this register is not implemented.

Field: Interface Version

Start of informative comment

This field contains the versions of the interface types defined in this specification. If a TPM implements both FIFO and CRB, the value in this register is the value of the active interface. If a TPM only implements one of the interface types, this register contains the version of that interface type. See Table 19 — FIFO Interface Identifier Register.

End of informative comment

1. If `TPM_INTERFACE_ID_x.InterfaceVersion` is 1111b, this field is invalid.
2. If `TPM_INTERFACE_ID_x.InterfaceVersion` is not 1111b:
 - a. A TPM SHALL report the version for the current active Interface type as indicated by the Interface Type field.
 - b. When FIFO is active, a TPM SHALL support `TPM_STS_x.commandCancel` and `TPM_STS_x.resetEstablishmentBit`.

Field: CapLocality

Start of informative comment

This field describes the interface capabilities of a TPM compliant to this specification. Some of the capabilities defined in this field may not be implemented in all TPMs. For example, a given Interface Type may support localities but not all implementations provide support for multiple localities. This field indicates which capability this implementation supports. This is a bit field where each bit is assigned to a capability.

Note: If a TPM supports multiple localities, it must support all the features of locality arbitration including Seize.

End of informative comment

1. If a TPM supports Locality 0 to 4, this field SHALL be set to 1.
2. If a TPM supports only Locality 0, this field SHALL be cleared to 0.
 - a. A TPM SHALL support all the fields and protocol for locality access.
 - b. A TPM SHALL NOT support TPM_ACCESS_x.Seize.

Field: CapFIFO

Start of informative comment

This state of this field indicates whether a TPM supports the FIFO interface or not. This field is read-only.

End of informative comment

1. If a TPM does not support the FIFO interface, this field SHALL be cleared to 0.
2. If a TPM supports the FIFO interface, this field SHALL be set to 1.
3. Writes to this field are ignored.

Field: CapCRB

Start of informative comment

This state of this field indicates whether a TPM supports the CRB interface or not. This field is read-only.

End of informative comment

1. If a TPM does not support the CRB interface, this field SHALL be cleared to 0.
2. If a TPM supports the CRB interface, this field SHALL be set to 1.
3. Writes to this field are ignored.

Field: InterfaceSelector

Start of informative comment

This field allows a caller to change the current TPM Interface. There are some requirements that must be met by any such implementation.

End of informative comment

1. This field MAY ONLY be changed if a TPM_INTERFACE_ID_x.CapFIFO and TPM_INTERFACE_ID_x.CapCRB are both set to 1 and TPM_INTERFACE_ID_x.IntfSelLock is cleared to 0.
2. Writes to this field SHALL be ignored if TPM_INTERFACE_ID_x.IntfSelLock is set to 1.
3. If a TPM_INTERFACE_ID_x.InterfaceType field is 0000b:
 - a. A write to this field of 00b SHOULD be ignored.
 - b. A write to this field of 01b SHALL change the active interface to CRB immediately following the next _TPM_INIT.
 - c. A TPM SHALL update a TPM_INTERFACE_ID_x.InterfaceType field to 0001b.
4. If a TPM_INTERFACE_ID_x.InterfaceType field is 0001b:
 - a. A write of 01b to this field SHOULD be ignored.
 - b. A write of 00b to this field SHALL change the active interface to FIFO immediately following the next _TPM_INIT.
 - c. A TPM SHALL update a TPM_INTERFACE_ID_x.InterfaceType field to 0000b.
5. If a TPM_INTERFACE_ID_x.InterfaceType field is 1111b, writes to this field SHALL be ignored.

Field: *IntfSelLock*

Start of informative comment

This field acts as a lock on a TPM_INTERFACE_ID_x.InterfaceSelector field. If this field is 1, writes to TPM_INTERFACE_ID_x.InterfaceSelector are ignored. This field is reset to 0 by a _TPM_INIT.

End of informative comment

1. This field MAY ONLY be changed if a TPM_INTERFACE_ID_x.CapFIFO and TPM_INTERFACE_ID_x.CapCRB are both set to 1.
2. Reads to this field SHALL return the correct value.
3. A write of 0 to this field SHALL be ignored.
4. A write of 1 to this field SHALL lock a TPM_INTERFACE_ID_x.InterfaceSelector field.
5. This field SHALL be cleared to 0 on _TPM_INIT.

Field: *CapSPICSUM*

Start of informative comment

This state of this field indicates whether a TPM supports the data checksum computation. If supported, a TPM computes the checksum on the entire command or response data. This field is read-only.

End of informative comment

1. Writes to this field SHALL be ignored.
2. If CapSPICSUM is set to 0b; Reads to TPM_DATA_CSUM_ENABLE and TPM_DATA_CSUM SHALL return 0xFF.
3. If CapSPICSUM is set to 1b:
 - a. A TPM SHALL support a TPM_DATA_CSUM_ENABLE register. See section 6.5.1.8.1 TPM_DATA_CSUM_ENABLE
 - b. A TPM SHALL support a TPM_DATA_CSUM register. See section 6.5.1.8.2 TPM_DATA_CSUM.

6.4.2.2 CRB Interface Identifier Register

Table 20 — CRB Interface Identifier Register

Abbreviation:			TPM_CRB_INTF_ID_x
General Description:			Interface Identifier Register
Default			Specific to each revision
Bit Descriptions:			
63:48	Read Only	DID	Device ID – vendor specific
47:32	Read Only	VID	Vendor ID- assigned by TCG. This is represented within the register in big-endian format. For example, a vendor ID of 0x1234 would be represented as: Bits 7:0 = 34 (0011 0100); Bits 15:8 = 12 (0001 0010).
31:24	Read Only	RID	Revision ID – specifies the revision of the component
23	Read Only	CapSPICSUM	0 – A TPM does not support calculation of a checksum 1 – A TPM supports calculation of a checksum for command and response. The capability is enabled via a TPM_DATA_CSUM_ENABLE register. See Section 6.5.1.8.1 TPM_DATA_CSUM_ENABLE
22:20	Read Only	Reserved	Reads return 0

19	Read Write	IntfSelLock	0 – A write of this value is ignored 1 – A write of this value locks the InterfaceSelector field and prevents further changes. Field is reset to 0 on _TPM_INIT
18:17	Read Write	InterfaceSelector	00 – A write of this value changes the selected interface to FIFO 01 – A write of this value changes the selected interface to CRB This field may only be written if IntfSelLock is 0. Writes to this field take effect on next _TPM_INIT. Other values are reserved.
16:15	Read Only	CapIFRes	Reserved for future interfaces (since InterfaceSelector is a two-bit field), reads return 0
14	Read Only	CapCRB	0 – CRB interface is not supported. 1 – CRB interface is supported and may be selected.
13	Read Only	CapFIFO	0 – FIFO interface is not supported. 1 – FIFO interface is supported and may be selected.
12:11	Read Only	CapDataXferSizeSupport	00 – TPM supports 4-byte transfer size only. 01 – TPM supports 8-byte transfer size (includes 4-byte transfers). 10 – TPM supports 32-byte transfer size (includes 4- and 8-byte transfers). 11 – TPM supports 64-byte transfer size (includes 4-, 8- and 32-byte transfers).
10	Read Only	Reserved	Reserved Reads return 0
9	Read Only	CapCRBIdleBypass	0 – TPM supports fast transition from Idle to Command Ready state 1 – TPM supports transition from Command Completion directly to Ready and supports transition from Command Completion directly to Command Reception. Note: if this field is set to 1, the driver may need to set the cmdReady bit twice if desiring to go to Ready, as there is no indicator that differentiates between Command Completion and Ready state in the CRB interface.
8	Read Only	CapLocality	0 – This interface supports Locality 0 only. Note: TPMs must support relinquishing Locality 0 so that no Locality is active. 1 – This interface supports 5 localities (Localities 0-4).
7:4	Read Only	InterfaceVersion	0010 – CRB interface version 1 with support for TPM Library 1.59. See Table 21 — CRB Historical Interface Versions 0000 – FIFO interface for TPM2.0.
3:0	Read Only	InterfaceType	0000 – FIFO interface as defined in PTP for TPM 2.0 is active. 0001 – CRB interface is active. 1111 – FIFO interface as defined in TIS1.3 is active (all other fields of this register are don't care).

Field: Interface Type

Start of informative comment

This field identifies the interface type currently active. While some TPMs may support either of these interfaces for product requirement and marketing purposes, only one interface can be active at a time. The selection of the interface is TPM Vendor Specific.

The state of this field governs the behavior of the rest of this register.

End of informative comment

1. A value of 1111b in this field SHALL be interpreted to mean a TPM supports only a PC Client TPM Interface Specification v1.3 [15] compliant FIFO interface.
2. If a TPM supports this specification, the value of this field SHALL NOT be 1111b.
3. Writes to this field SHALL be ignored.
4. If this field is set to 0000b:
 - a. A TPM SHALL correctly report all other capabilities for TPM_CRB_INTF_ID_x fields.
 - b. A TPM SHALL support TPM_CRB_INTF_ID_x.InterfaceVersion, which SHALL be 0000b for the FIFO interface.
 - c. A TPM MAY support TPM_INTF_CAPABILITY_x.InterfaceVersion
5. If this field is set to 0001b:
 - a. A TPM SHALL correctly report all other capabilities for TPM_CRB_INTF_ID_x fields.
 - b. A TPM SHALL support TPM_CRB_INTF_ID_x.InterfaceVersion, which SHALL be 0010b for the CRB interface.
 - c. A TPM SHALL NOT support TPM_INTF_CAPABILITY_x.InterfaceVersion.
6. If this field is set to 1111b, this register is not implemented.

Field: Interface Version

Start of informative comment

This field contains the versions for the interface types defined in this specification. If a TPM implements both CRB and FIFO, the value in this register is the value of the active interface. If a TPM only implements one of the interface types, this register contains the version of that interface type. See Table 20 — CRB Interface Identifier Register.

The historical versions of the CRB interface are defined as follows:

Table 21 — CRB Historical Interface Versions

Interface Version (Decimal)	Interface Version (Hex)	Description
0	0000	CRB Versions prior to PTP standardization of the interface
1	0001	First version of CRB standardized in the PTP
2	0010	Increased CRB Data Buffer size to accommodate additional CRT coefficients in the RSA Private structure for TPM2_Create added in TPM Library 1.59

End of informative comment

1. If TPM_CRB_INTF_ID_x.InterfaceVersion is 1111b, this field is invalid.
2. If TPM_CRB_INTF_ID_x.InterfaceVersion is not 1111b:
 - a. A TPM SHALL report the version for the current active Interface type as indicated by the Interface Type field.
 - b. When FIFO is active, a TPM SHALL support TPM_STS_x.commandCancel and TPM_STS_x.resetEstablishmentBit.

Field: CapLocality

Start of informative comment

This field describes the interface capabilities of a TPM compliant to this specification. Some of the capabilities defined in this field may not be implemented in all TPMs. For example, a given Interface Type may support localities but not all implementations provide support for multiple localities. This field indicates which capability this implementation supports. This is a bit field where each bit is assigned to a capability.

End of informative comment

1. If a TPM supports Locality 0 to 4, this field SHALL be set to 1.
2. If a TPM supports only Locality 0, this field SHALL be cleared to 0.
 - a. A TPM SHALL support all the fields and protocol for locality access.
 - b. A TPM SHALL NOT support TPM_LOC_CTRL_x.Seize.

Field: CapCRBIdleBypass

Start of informative comment

This field describes the behavior of a TPM in transitioning states from Command Completion to Ready and from Command Completion to Command Reception. If this field is set to a 0, a TPM will transition from Command Completion to Idle and on receiving a write of 1 to cmdReady, will transition to Ready. A TPM with this field set 1 will accept a write of 1 to cmdReady from Command Completion to bypass the Idle state. A TPM with this field set to 1 will also accept data written to the data buffer, bypassing the Idle and Ready states, and going directly to Command Reception. Because there is no differentiating indicator in the interface between Command Completion and Ready, a driver will need to keep track of the state in which it leaves a TPM. It is expected that drivers will only exercise this bypass mechanism when multiple commands are queued up for a TPM, as a TPM may perform necessary background processing while in Idle. Drivers should take care when using this capability as TPMs require the Idle state to perform clean-up operations, e.g., freeing up NVRAM or entering low power mode. TPMs that provide this capability need to account for the possibility they may not enter Idle for extended periods of time.

End of informative comment

1. If a TPM supports transitions from Command Completion to Ready and from Command Completion to Command Reception, this field SHALL be set to 1.
2. If a TPM supports fast transition from Idle to Ready, this field SHALL be set to 0.

Field: CapDataXferSizeSupport

Start of informative comment

This field is supported only in the CRB specific Interface Identifier Register. It provides the information added to the PC Client TPM Interface Specification for TPM 1.2 v 1.3 [15] to support larger transfer sizes on the SPI bus.

The state of this field indicates the size of transfers supported by a TPM on the hardware interface.

End of informative comment

Field: CapFIFO

Start of informative comment

This state of this field indicates whether a TPM supports the FIFO interface or not. This field is read-only.

End of informative comment

1. If a TPM does not support the FIFO interface, this field SHALL be cleared to 0.
2. If a TPM supports the FIFO interface, this field SHALL be set to 1.
3. Writes to this field are ignored.

Field: CapCRB

Start of informative comment

This state of this field indicates whether a TPM supports the CRB interface or not. This field is read-only.

End of informative comment

1. If a TPM does not support the CRB interface, this field SHALL be cleared to 0.
2. If a TPM supports the CRB interface, this field SHALL be set to 1.
3. Writes to this field are ignored.

Field: InterfaceSelector

Start of informative comment

This field allows a caller to change the current TPM Interface. There are some requirements that must be met by any such implementation.

End of informative comment

1. This field MAY ONLY be changed if a TPM_CRB_INTF_ID_x.CapFIFO and TPM_CRB_INTF_ID_x.CapCRB are both set to 1 and TPM_CRB_INTF_ID_x.IntfSelLock is cleared to 0.
2. Writes to this field SHALL be ignored if TPM_CRB_INTF_ID_x.IntfSelLock is set to 1.
3. If a TPM_CRB_INTF_ID_x.InterfaceType field is 0000b, then:
 - a. A write to this field of 00 SHOULD be ignored.
 - b. A write to this field of 01 SHALL change the active interface to CRB immediately following the next _TPM_INIT.
 - c. A TPM SHALL update a TPM_CRB_INTF_ID_x.InterfaceType field to 0001b.
4. If a TPM_CRB_INTF_ID_x.InterfaceType field is 0001b, then:
 - a. A write of 01b to this field SHOULD be ignored.
 - b. A write of 00b to this field SHALL change the active interface to FIFO immediately following the next _TPM_INIT.
 - c. A TPM SHALL update a TPM_CRB_INTF_ID_x.InterfaceType field to 0000b.
5. If a TPM_CRB_INTF_ID_x.InterfaceType field is 1111b, writes to this field SHALL be ignored.

Field: IntfSelLock

Start of informative comment

This field acts as a lock on a TPM_CRB_INTF_ID_x.InterfaceSelector field. If this field is 1, writes to TPM_CRB_INTF_ID_x.InterfaceSelector are ignored. This field is reset to 0 by a _TPM_INIT.

End of informative comment

1. This field MAY ONLY be changed if a TPM_CRB_INTF_ID_x.CapFIFO and TPM_CRB_INTF_ID_x.CapCRB are both set to 1.
2. Reads to this field SHALL return the correct value.
3. A write of 0 to this field SHALL be ignored.
4. A write of 1 to this field SHALL lock a TPM_CRB_INTF_ID_x.InterfaceSelector field.
5. This field SHALL be cleared to 0 on _TPM_INIT.

6.5 TPM's Software Interaction

Start of informative comment

When a platform is powered on, platform hardware issues a _TPM_INIT to a TPM. After each _TPM_INIT, the platform must issue a TPM2_Startup command to a TPM before issuing any other TPM command, except for the HASH_START/_DATA/_END interface commands described in Section 5.3 Locality-Controlled Functions. The command and startup type inform a TPM how to initialize itself, for example, by informing a TPM to restore or clear the state of the PCRs that may retain their state across an S3 suspend. The platform firmware is required to perform

a TPM2_Startup command. With respect to locality, it is important to understand that the locality using a TPM and its interface is architected to be a non-preemptive use of a TPM. When there are multiple Software users spanning multiple localities, the following explains the handshake mechanism.

Each Software agent, when it wishes to use a TPM, must request use of the locality it wishes to access. If a TPM is idle, the first agent that sets this field will become the user. A TPM must set active locality to the locality that gains access to TPM. All other localities that have requested use of a TPM must poll on a TPM_ACCESS_x or a TPM_LOC_STS_x register to determine when they are granted access to a TPM.

When the currently active locality is finished with a TPM, it will relinquish locality. A TPM will look at all pending requests to use a TPM and grant the access to the highest locality with a pending request.

If Software, for some reason, decides a lesser locality's Software is not playing fair or is hung (by exceeding the maximum timeout value as specified by the TSS), then it can seize a TPM from the current user if the current user is at a lower locality. This forces a TPM to stop honoring cycles from the current locality, and only honor the new locality's requests.

End of informative comment

6.5.1 Interface-Agnostic functions

Start of informative comment

Command aborts are interface dependent and are defined within the Interface specific sections of this specification.

End of informative comment

6.5.1.1 Bus Aborts

Start of informative comment

For LPC, an LPC Abort cycle requires that the cycle have no effect on a TPM. There are two possible implementations, either of which is acceptable:

1. A TPM may simply not drive a valid LPC SYNC. This will cause the chipset to return FFh to the CPU for reads. The write data will be dropped.
2. A TPM may accept writes and do nothing with them. A TPM will not provide any TPM-Response to these writes, nor does it provide any indication that it has seen a write but dropped it. For reads, a TPM, if it responds with a valid LPC SYNC, must return FFh as the data. This mimics a true LPC or PCI Master Abort from the CPU's perspective.

For SPI, there is no analog to the LPC SYNC signal. To provide a similar mechanism to the LPC Abort cycle, this specification defines a protocol using MISO.

End of informative comment

LPC Aborts:

1. An LPC Abort cycle SHALL cause a TPM to ignore the current LPC bus cycle.

SPI Aborts:

1. For Read Cycles, a TPM SHALL abort a cycle by driving 1 on MISO and continue to hold MISO at 1 until its CS# signal is deasserted.
2. For Write Cycles, a TPM SHALL abort a cycle by driving a 1 on MISO, then drop all incoming data.
3. A TPM MAY use the standard SPI Wait mechanism, as defined in Section 7.4.5 Flow Control, to insert a wait state while decoding the cycle before issuing an abort.

6.5.1.2 Failure Mode

Start of informative comment

Several conditions can cause a TPM to enter a specifically defined state called “failure mode”. These conditions and the behavior of a TPM are defined in a TPM2 Library Specification [7] and are not reproduced here. This section defines additional considerations specific to the behavior of a TPM Interface.

The new Normative 2 below allows the system Software to perform the allowed remediation actions on a TPM. The requirement to allow changing locality exists because a TPM’s locality when it entered failure mode may not be the appropriate locality for performing the allowed remediation.

End of informative comment

While a TPM is in a failure mode:

1. In addition to the requirements called out in a TPM2 Library Specification [7], the HASH_START, HASH_DATA and HASH_END interface commands SHALL appear to the caller as dropped writes (i.e., they are not allowed to hang or suspend the system), but SHALL NOT perform any actions on a TPM such as those specified in Section 5.3.1 DRTM Execution Sequence.
2. All FIFO registers SHALL remain fully functional including the ability to change locality.
3. All CRB registers SHALL remain fully functional including the ability to change locality.

6.5.1.3 Command Duration

Start of informative comment

It is important to distinguish between the two terms: duration and timeout. Duration is the amount of time for a TPM to execute a command once a TPM has received the command’s complete set of bytes and the Software starts the operation by writing a 1 to TPM_STS_x.tpmGo or TPM_CRB_CTRL_x.Start. Duration has no relationship to the timings of the interface protocols. Timeouts, referenced below, are not related to the interface timeouts, as defined in Section 6.5.1.4 Interface Timeouts. The timeouts defined in Table 22 —Command Timing below, are defined to allow driver writers to know when to issue a command cancel to attempt to recover a TPM. If a TPM fails to complete the command within the timeout defined in Table 22, the driver may safely assume a TPM has had a critical failure and is non-recoverable. The duration and timeout for any command performing a TPM2_NV_Read are defined for the pre-OS environment only. Some TPM implementations may rely on OS drivers to access non-volatile memory in an OS-present environment, and as such may not be able to comply with these timings.

During a platform’s early boot phase, performance is critical, and resources are limited. For this reason, constraints are placed on commands that are typically required during this phase, to eliminate the need to compensate for implementation differences between different TPMs. Since the same platform requirements drive the reasons for making commands available before a self-test has completed, the commands listed in this section are similar to those in the Section 6.5.1.6 Self-Test and Early Platform Initialization.

Note: TPMs that comply with the FIPS 140-3 Self-Test Requirements may exceed the Command Duration for TPM2_SelfTest (FULLTEST = yes) if larger asymmetric key sizes are supported, e.g., RSA 4096.

End of informative comment

1. Command Duration is defined as the time between a TPM’s receipt of a 1 to TPM_STS_x.tpmGo or TPM_CRB_CTRL_x.Start and a TPM completing the command as evidenced by:
 - a. FIFO Interface: a TPM sets both TPM_STS_x.dataAvail and TPM_STS_x.stsValid fields to a 1.
 - b. CRB Interface: a TPM clears TPM_CRB_CTRL_x.Start to 0.
2. Command Duration for the _TPM_Hash_X commands is defined as the time between a TPM’s receipt of the address and completion of the command, as evidenced by CS# deassertion for SPI and an I2C STOP command. Note: The Command Duration for HASH_DATA is defined based on a write of a 64B block of data to a TPM with a single bank enabled.
3. For the commands listed in Table 22, a TPM SHOULD not exceed the Duration values and SHALL NOT exceed the Timeout values.

Note: The timings in Table 22 assume that any algorithms required for these functions do not need to be tested during the execution of the command, see Section 6.5.1.6 Self-Test and Early Platform Initialization. Additionally,

the duration and timeout for TPM2_VerifySignature do not apply to usage of TPM2_VerifySignature with RSA 3072-bit keys. Additional time may be expected for usage of TPM2_VerifySignature with RSA 3072-bit keys.

Table 22 —Command Timing

Commands	Duration [ms]	Timeout [ms]
Signals		
_TPM_Hash_Start	20	750
_TPM_Hash_Data	20	750
_TPM_Hash_End	20	750
Startup		
TPM2_Startup	20	750
Testing		
TPM2_SelfTest(fullTest=YES)	1000	3000
TPM2_SelfTest(fullTest=NO)	20	750
Random Number Generator		
TPM2_GetRandom	750	2000
Hash/HMAC/Event Sequences		
TPM2_HashSequenceStart	20	750
TPM2_SequenceUpdate	20	750
TPM2_SequenceComplete	20	750
TPM2_EventSequenceComplete	20	750
Signature Verification		
TPM2_VerifySignature	750	2000
Integrity Collection (PCR)		
TPM2_PCR_Extend	20	750
Hierarchy Commands		
TPM2_HierarchyControl	750	2000
TPM2_HierarchyChangeAuth	750	2000
Capability Commands		
TPM2_GetCapability	20	750
Non-volatile Storage		
TPM2_NV_Read	750	2000

6.5.1.4 Interface Timeouts

Start of informative comment

The term timeout, in this section, applies to timings between various states or transitions within the interface protocol. Interface timeout values are not related to duration (see Section 6.5.1.3 Command Duration).

Because of the variations between implementations, it is not practical to specify timeout values that apply to all implementations. Efficient Software must have the means to provide optimizations; therefore, Software should be able to determine, with some level of granularity, when a state transition is expected to complete and when the Software should determine that a TPM has failed. These timings are called timeouts. The timeouts have been broken into four categories, each with a label.

This specification defines a maximum value for each of these timeouts as defined in Table 23 below.

Note: TIMEOUT_B applies to the time it takes to cancel a command. As canceling a command affects more than the interface, this is the only timeout that applies to higher levels of a TPM stack.

End of informative comment

1. There are four timeout values designated: TIMEOUT_A, TIMEOUT_B, TIMEOUT_C, and TIMEOUT_D.
2. The values for the timeouts SHALL be as shown in Table 23.

Table 23 — Definition of Interface Timeouts

TIMEOUT Label	Default Timeouts
TIMEOUT_A	750 milliseconds
TIMEOUT_B	2,000 milliseconds
TIMEOUT_C	200 milliseconds
TIMEOUT_D	30 milliseconds

6.5.1.5 _TPM_INIT and Reset

Start of informative comment

The command _TPM_INIT is not an actual command with a defined ordinal and set of parameters; rather, it is an indication to a TPM that the Static RTM is being reset and that the RTR and RTS should also be reset. On a PC Client with a physical TPM, this is performed using a hardware-based signal. On a PC Client with an integrated TPM, the integrated TPM receives a vendor specific indication that a TPM should come out of reset.

Note: This distinction is made because it is conceivable that other architectures might use other methods for performing this function.

For a TPM implementation using a TPM Packaging specified in Section 9.1 _TPM_INIT. _TPM_INIT is indicated by the transition of LRESET# or SPI_RST# pin from low to high. There is no requirement to use this packaging; therefore, it is up to a TPM manufacturer to define the hardware-based signal that performs this function.

Warning: Some TPMs accept commands while their reset pin is reasserted after _TPM_INIT. TPM state changes that occur in this case would be cleared when the reset pin is deasserted. Motherboard and platform firmware designers should note that the proper way to disable a TPM is to send a TPM_Startup (CLEAR), followed by a TPM2_Hierarchy_Control to disable all the TPM's hierarchies. See the TCG PC Client Platform Firmware Profile for TPM 2.0 Systems, Section TPM Visibility to the OS for the procedure to disable the TPM and hide it from an OS.

End of informative comment

1. For TPMs with dedicated physical packages:
 - a. A TPM SHALL implement a hardware-based signal for _TPM_INIT.
 - i. If a TPM uses a TPM Packaging specified in Section 9.1 TPM Packaging, the _TPM_INIT signal shall be assigned to the reset pin (LRESET# for LPC and I2C, SPI_RST# for SPI).
 - ii. If a TPM does not use a TPM Packaging specified in Section 9.1 TPM Packaging, a TPM Manufacturer SHALL define the pin used for _TPM_INIT.
 - b. A TPM SHALL be reset on the deassertion of pin used for _TPM_INIT.
 - i. A TPM SHOULD NOT accept commands when the reset PIN is asserted.
2. For TPMs without dedicated physical packages:
 - a. A TPM SHOULD be integrated so that a TPM Reset is deasserted when the Static RTM Reset is deasserted.

6.5.1.6 Self-Test and Early Platform Initialization

Start of informative comment

During the time-sensitive phase of a PC Client's startup procedure, only a small subset of the available commands is likely to be necessary. Therefore, this specification requires a TPM to perform any necessary self-test on these commands required to make them available upon completion of `_TPM_INIT`. Note that some of the commands in this list require use of the hash function and the random number generator and therefore these functions are tested as part of power on self-test.

This specification defines the maximum time a TPM can take to complete its self-test time. Due to variations in security requirements and implementations of TPM, it is difficult to mandate this to the satisfaction of all TPM vendors for all PC Client implementations. However, the generally accepted constraints of this platform's architecture and applications target the 1 to 2 second timeframe. PC Client platform manufacturers are advised to keep this aspect of a TPM's specification in mind when selecting a TPM for applicability to the platform's targeted use.

Graphically, the initialization sequence is as follows:

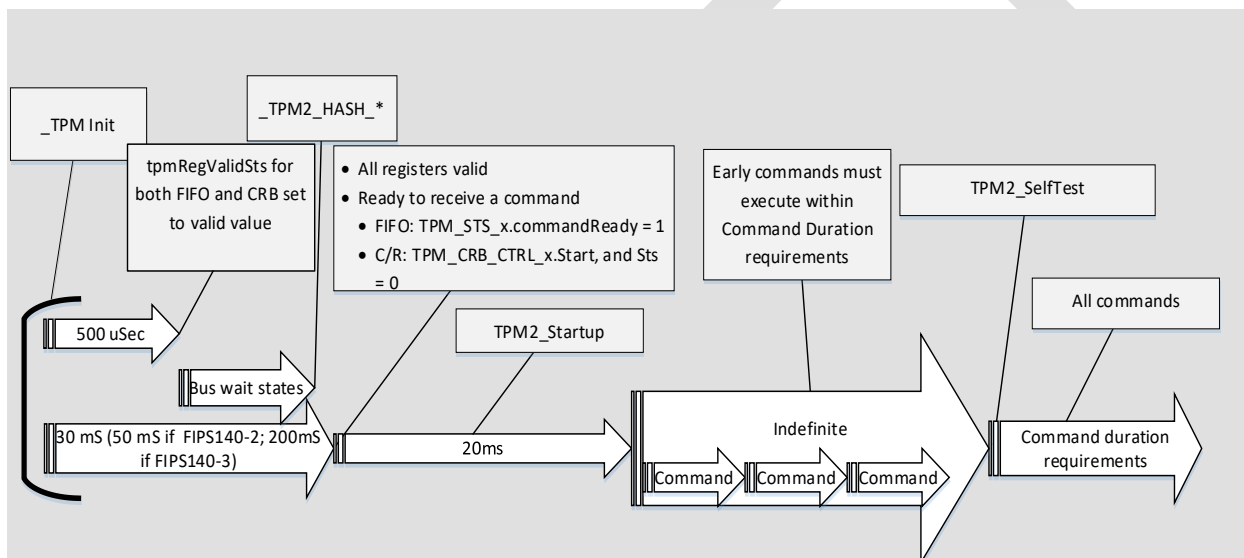


Figure 2 — PC Client Initialization Sequence

A TPM is required to recognize the deassertion of its reset signal, but the specification does not require a TPM to recognize or respond to the assertion of reset. This allows a TPM to perform internal initialization and test steps to improve boot performance. The PFP requires a TPM to be reset at the same time as the host CPU.

A TPM might respond to commands while reset is asserted. See Section 6.5.1.5 `_TPM_INIT` and Reset.

End of informative comment

1. After `_TPM_INIT` and prior to `TPM2_Startup`, a TPM SHALL test all internal functions that are necessary to perform the following commands:
 - a. `_TPM_Hash_Start/ _Data/ _End`
 - b. `TPM2_Startup`
2. After `_TPM_INIT`, a TPM SHALL test all internal functions that are necessary to perform the following commands necessary for early boot operations. The following operations SHALL be available after `TPM2_Startup` and before a call to `TPM2_SelfTest` and SHALL NOT cause an implicit self-test.
 - a. `TPM2_HashSequenceStart`
 - b. `TPM2_SequenceUpdate`
 - c. `TPM2_EventSequenceComplete`
 - d. `TPM2_SequenceComplete`

- e. TPM2_PCR_Extend
 - f. TPM2_SelfTest
 - g. TPM2_GetRandom
 - h. TPM2_HierarchyControl
 - i. TPM2_HierarchyChangeAuth
 - j. TPM2_SetPrimaryPolicy
 - k. TPM2_GetCapability
 - l. TPM2_NV_Read
3. The maximum time to continue TPM self-test after receipt of TPM2_SelfTest (fullTest = NO) SHOULD be less than 1 second.

6.5.1.7 Data Buffer Size

Start of informative comment

Software must be aware of the maximum amount of data it can transfer to a TPM in one command. This is most relevant to the NV Storage functions where relatively large amounts of storage area can be defined by the Software. The size of a TPM data buffer does not prevent larger areas from being defined. It means, however, that if an area is defined that requires more data than the input buffer can accommodate in one transaction, the Software must break up the write into smaller pieces. This is possible because the NV Write commands allow for an offset.

Note that there is no specified output buffer size. A TPM may leverage the same buffer for command and response data. If TPM has separate buffers, a TPM will return an error on the command before exceeding its output buffer size.

TPMs can provide a larger input buffer size to support increased performance.

TPMs implemented with the CRB address map as defined in this specification are limited to a data buffer size of 3968 bytes. This maximum size prevents the data buffer from crossing a locality address boundary.

End of informative comment

A TPM SHALL support a data buffer size large enough to support the largest implemented command.

6.5.1.8 Command Response Checksum

Start of informative comment

The checksum of the received command or the checksum of the response returned by a TPM can be computed in a TPM_DATA_CSUM register. If a TPM support this feature, the capability must be indicated in the TPM_INTERFACE_ID register for the FIFO interface or in the TPM_CRB_INTF_ID register for the CRB interface. This feature can be activated through the TPM_DATA_CSUM_ENABLE register. When the CSUM feature is activated, a TPM returns the entire command data checksum in a TPM_DATA_CSUM register at the end of the command transmission and returns the entire response data checksum at the end of the response transmission. If no locality is active, the TPM_DATA_CSUM register should return 0xFF and the TPM_DATA_CSUM_ENABLE register should accept a write/read access. If another locality is active, the TPM_DATA_CSUM register should return 0xFF and the TPM_DATA_CSUM_ENABLE register should accept a write/read access, see Table 47 — Register Behavior Based on Locality Setting for FIFO and Table 48 — Register Behavior Based on Locality Setting for CRB.

End of informative comment

1. A TPM SHALL use CRC-CCITT (KERMIT) for the calculation of the data checksum (see Section 9 for further details). The KERMIT parameters are as follows:
 - a. The generator polynomial is $0x1021 (x^{16} + x^{12} + x^5 + 1)$
 - b. The initialization value is 0x0000.
 - c. Reflection of input data: TRUE
 - d. Reflection of output data: TRUE
 - e. Final XOR: 0x0000
2. Test vectors:

- a. The CRC value for the ASCII string “123456789” is 0x8921.
A 2-byte read from 0x44 will return
Data0 (LSB) = 0x21 and Data1 (MSB) = 0x89.
 - b. The CRC value for the ASCII string “1122334455” is 0xD367.
A 2-byte read from 0x44 will return
Data0 (LSB) = 0x67 and Data1 (MSB) = 0xD3.
 - c. The CRC value for the HEX string 00 C1 00 00 00 0C 00 00 00 99 00 01₁₆
(TPM2_StartUp(ST_CLEAR)) is 0xFBBF.
A 2-byte read from 0x44 will return
Data0 (LSB) = 0xBF and Data1 (MSB) = 0xFB.
 - d. The CRC value for the HEX string 80 01 00 00 00 0C 00 00 01 44 00 00₁₆
(TPM2_StartUp(TPM_SU_CLEAR)) is 0x6733.
A 2-byte read from 0x44 will return
Data0 (LSB) = 0x33 and Data1 (MSB) = 0x67.
3. If enabled via a TPM_DATA_CSUM_ENABLE register:
- a. A TPM SHALL calculate a checksum over the entire command.
 - b. A TPM SHALL calculate a checksum over the entire response.
 - c. A TPM SHALL update the command checksum before the transition of TPM_STS.Expect from 1 to 0 in FIFO mode. In CRB mode a TPM SHALL update the command checksum after reception of the last command byte and shall set a TPM_CRB_CTRL_STS.cSUMAvailable bit to 1 when the checksum of the command is computed.
 - d. A TPM SHALL maintain the command checksum from the transition of TPM_STS.Expect from 1 to 0 until TPM_STS.tpmGo is set to 1 in FIFO mode. In CRB mode TPM SHALL maintain the command checksum from the transition of TPM_CRB_CTRL_STS.cSUMAvailable until TPM_CRB_CTRL_START.Start is set to 1. TPM_CRB_CTRL_STS.cSUMAvailable SHALL be cleared by a TPM when the host writes a 1 to TPM_CRB_CTRL_REQ.cmdReady or after reception of the first byte.
 - e. A TPM SHALL update the response checksum before the transition of TPM_STS.dataAvail from 0 to 1 in FIFO mode. In CRB mode a TPM SHALL update the response checksum before the transition of TPM_CRB_CTRL_START.Start from 1 to 0.
 - f. A TPM SHALL maintain the response checksum from the transition of TPM_STS.dataAvail from 0 to 1 until the host writes a 1 to TPM_STS.commandReady in FIFO mode. In CRB mode a TPM SHALL maintain the response checksum until the host writes a 1 to TPM_CRB_CTRL_REQ.cmdReady.

6.5.1.8.1 TPM_DATA_CSUM_ENABLE

Start of informative comment

This register may be volatile and may not be preserved across a TPM_RESET or a TPM_RESTART. Device drivers may need to check this field and re-enable it.

End of informative comment

Table 24 — TPM_DATA_CSUM_ENABLE Definition

Abbreviation:		TPM_DATA_CSUM_ENABLE	
General Description:		Enables the data checksum calculation and indication via a TPM_DATA_CSUM register.	
Bit Descriptions:			
31:1		Reserved	Reads always return 0
0	Read/ Write	dataCSumEnable	1 = Data Checksum enabled 0 = Data Checksum disabled (default)

6.5.1.8.2 TPM_DATA_CSUM**Start of informative comment**

This register is updated after each command received and before each response returned by a TPM. When the last byte of the command is received, a TPM computes the checksum over all data received and updates the TPM_DATA_CSUM register before clearing the TPM_STS.Expect bit for the FIFO interface or before setting the cSUMAvailable bit in the TPM_CRB_CTRL_STS register for the CRB interface. A TPM must compute the checksum of the entire data of the response before setting TPM_STS.dataAvail for the FIFO interface or before clearing the TPM_CRB_CTRL_START register.

End of informative comment**Table 25 — TPM_DATA_CSUM Definition**

Abbreviation:		TPM_DATA_CSUM	
General Description:		Contains the data checksum when enabled	
Bit Descriptions:			
31:16		Reserved	Reads always return 0
15:0	Read only	dataChecksum	Read returns the checksum of the entire command data at the end of the command transmission or the checksum of the entire response data at the end of the response transmission. Default: 0x00

6.5.1.9 Errors**Start of informative comment**

In general, there are four types of errors for a TPM, as outlined in the following cases:

- 1) Errors that a TPM detects and understands and that force it into Failure Mode:
 - a) In this mode, a TPM responds correctly to all register reads or writes.
 - b) A TPM provides a TPM-defined Response to security operations. This response should be one of the error return codes defined in a TPM Library Specification, e.g. TPM2_RC_FAILURE.
 - c) A TPM allows certain TPM-defined transactions, e.g. TPM2_GetTestResult, to return a response that indicates the particular error, or provides other TPM status information.
- 2) Errors that a TPM detects and that seem to be attacks on the hardware interface:
 - a) A TPM completely stops responding and enters Shutdown because of these events.

- b) A TPM may not respond to reads or writes of the physical interface.
- c) If the CRB interface is implemented, a TPM may return a default TPM_CRB_CTRL_x.Status value of 0001h.
- d) A TPM may not provide any response.
- e) All accesses to a TPM in this state should result in a bus Abort (as defined in Section 6.5.1.1 Bus Aborts) until a TPM reset has occurred.

3) Transmission or protocol errors:

- a) TPM registers and Software behavior have been defined to both identify and correct overruns or underruns on both reads and writes.
- b) Errors that a TPM does not detect but cause it to hang or shutdown.

4) Errors in which a TPM is unresponsive.

For case 1, Failure Mode, there is no need for a status field or interrupt, since the Response contains all the information that Software needs to understand a TPM state. Case 1 may include things such as the RNG self-test failed.

For case 2, Shutdown, on a FIFO interface, there is no need for a status field or interrupt since an LPC TPM does not respond to any cycles at all and an SPI TPM aborts all cycles, as defined in Section 6.5.1.1 Bus Aborts. Reading FFh from TPM_ACCESS_x indicates this state. In addition to the behavior defined by the physical interface, the CRB interface provides for a TPM to respond to reads of the Status field with a default value. TPMs may respond with this default value or may timeout on the read request.

For case 3, the interface has been defined with the status fields needed to detect overruns and underruns and provides a mechanism to recover from those errors if they are transient. Software should time out after some number of retries.

Case 4 obviously needs no status field or other indication. Note that Software may be able to determine that a TPM is in a hung or error state, even though a TPM cannot. For instance, if a TPM hangs in a microcode loop, then status fields would never be updated. Software could detect this case if it has sent a command and a TPM's registers are not updated within the required timeout.

The requirement for all errors is that system security or secrets cannot be compromised.

Therefore, this specification lumps all errors relating to a TPM into one of the first three categories. For instance, assume there is a long power glitch. A TPM can treat this like an attack, so LPC TPMs simply cease operation while SPI TPMs bus abort all cycles, or it could go to the Error Mode and return error responses to all commands. The exact condition that forces a TPM into one error state or the other is vendor specific.

This specification defines a TPM's behavior for protocol or transmission errors. It is beyond the scope of this specification to define every hardware or Software attack. It is within the scope of this specification to define the protocol for dealing with the errors.

If a TPM is in a state as defined in cases 1-3 above, or in the working state, it meets the requirements of this specification. A TPM must never allow secrets to be disclosed or respond with anything other than an error. It may use any of the above cases to enforce this rule.

As only the transmission errors are taken care of by the transmission protocol, a TPM has just two options for other errors: go into Failure Mode or to Shutdown Mode. Since each vendor's TPM will have different physical implementations, there is no way to precisely define each error and whether it should cause a TPM to go into Failure Mode or Shutdown. Therefore, it is vendor specific whether an error causes Failure Mode or Shutdown Mode responses. A TPM Library Specification [7] prescribes what responses a TPM must return when in Failure Mode.

Software has two cases to deal with. If a TPM has shutdown, this is detected by TPM registers not being updated within the appropriate timeout, see Section 6.5.1.4 Timeouts, and the platform should be rebooted. If a TPM returns Error Responses, then the Software should already be designed to handle this case.

Since the recovery for Shutdown Mode is system reset and the recovery for Failure Mode is also system reset, there is no need to define in more detail how a TPM should handle each error. In the future, if an error has a more graceful recovery mechanism, then the specification will need to be more precise on how a TPM must handle the error. Today,

the Software stack will simply detect a timeout in the protocol and reset the system or it will detect that a TPM is only returning Error Responses for TPM commands and reset the system.

Note: Software attacks should never cause a TPM to enter Shutdown Mode. Shutdown Mode is intended to counter hardware attacks and should not persist through a `_TPM_INIT` unless the hardware attack also persists.

End of informative comment

1. In the event of any error, a TPM SHALL NOT compromise system security or secrets.
2. If a TPM detects an error:
 - a. A TPM SHALL respond correctly to Register Reads and Writes, i.e., it must either correctly read/write the register or perform a bus abort; but it SHALL not hang the bus.
 - b. A TPM MAY respond with a defined TPM Error Response to Security Operations.
 - c. A TPM MAY respond by entering Failure mode.
3. If a TPM detects a hardware attack,
 - a. A TPM SHALL enter Shutdown Mode until a subsequent `_TPM_INIT`
 - b. A TPM MAY respond to a read of `TPM_CRB_CTRL_x.Status` with a value of `0001h`.
4. Following `_TPM_INIT`, a TPM SHALL clear Shutdown Mode or Failure Mode unless the attack or error condition remains.

6.5.2 FIFO Interface Requirements

6.5.2.1 FIFO Register Space Addresses

Start of informative comment

Table 26 lists the addresses decoded by a TPM when a FIFO is implemented. A `TPM_ACCESS_x` register has multiple, separate, and unique instances, one per locality. The other register addresses alias to a single register with the locality used to determine if accesses are permitted or aborted as defined in Section 6.5.2.3.1 Command Aborts.

End of informative comment

Table 26 — Allocation of Register Space for FIFO TPM Access

Offset	Register Name	Description
Locality 0		
0000h	TPM_ACCESS_0	Used to gain ownership of a TPM for this locality.
0007h-0001h	Reserved	Reserved for future use.
000Bh-0008h	TPM_INT_ENABLE_0	Interrupt configuration register
000Ch	TPM_INT_VECTOR_0	SIRQ vector to be used by a TPM
000Dh-000Dh	Reserved	Reserved for future use.
000Fh-000Eh	Reserved	Reserved for future use.
0013h-0010h	TPM_INT_STATUS_0	Shows which interrupt has occurred
0017h-0014h	TPM_INTF_CAPABILITY_0	Provides information about supported interrupts and the characteristic of the burstCount register of the TPM.
001Bh-0018h	TPM_STS_0	Status Register. Provides status of a TPM
0023h-001Ch	Reserved	Reserved for future use.
00027h-0024h	TPM_DATA_FIFO_0	ReadFIFO or WriteFIFO, depending on the current bus cycle (read or write). These four addresses are aliased to one inside a TPM. Note: A TPM is not required to check that the addresses on the LPC bus are incrementing modulo-4, even though platform hardware would most likely send it that way. The read or write data could be performed by accessing 0024h repeatedly without using the other addresses.
002Fh-0028h	Reserved	Reserved for future use.

Offset	Register Name	Description
0033h-0030h	TPM_INTERFACE_ID_0	Provides information on the interface type(s) supported by a TPM. This register is aliased across localities
0038h-0034h	TPM_DATA_CSUM_ENABLE_0	Register to enable computed checksum on command and response
003Bh-0038h	TPM_DATA_CSUM_0	Register to read the checksum computed
007Fh-003Bh	Reserved	Reserved for future use.
0083h-0080h	TPM_XDATA_FIFO_0	Extended ReadFIFO or WriteFIFO, depending on the current bus cycle (read or write). Transactions to this address may be any size from 1 byte to the size reported in TPM_INTF_Capability_x.DataTransferSizeSupport (see section 6.5.2.7 Interface Capability). A TPM MAY alias this address with a TPM_DATA_FIFO at offset 0x0024. Note: Aliasing this register with a TPM_DATA_FIFO allows for vendor optimization by sharing the same internal data buffer.
00BFh-0084h	Reserved	Reserved. These addresses are reserved by the chipset. A TPM should not respond to accesses to these addresses. Reserving this address range ensures that Software which issues writes larger than 1 byte to offset 0x0080 doesn't inadvertently encounter a register in a TPM in this space.
0EFFh-00C0h	Reserved	Reserved for future use.
0F03h-0F00h	TPM_DID_VID_0	Vendor and device ID
0F04h	TPM_RID_0	Revision ID
0F7Fh-0F05h	Reserved	Reserved for future use.
0F8Fh-0F80h	Reserved	Reserved for future use.
0FFFh-0F90h		Vendor-defined configuration registers
Locality 1		
1000h	TPM_ACCESS_1	Used to gain ownership of a TPM for this locality.
1007h-1001h	Reserved	Reserved for future use.
100Bh-1008h	TPM_INT_ENABLE_1	Same as TPM_INT_ENABLE_0
100Ch	TPM_INT_VECTOR_1	Same as TPM_INT_VECTOR_0
100Fh-100Dh	Reserved	Reserved for future use.
1013h-1010h	TPM_INT_STATUS_1	Same as TPM_INT_STATUS_0
1017h-1014h	TPM_INTF_CAPABILITY_1	Same as TPM_INTF_CAPABILITY_0
101Bh-1018h	TPM_STS_1	Same as TPM_STS_0
1023h-101Ch	Reserved	Reserved for future use.
1027h-1024h	TPM_DATA_FIFO_1	Same as TPM_DATA_FIFO_0
102Fh-1028h	Reserved	Reserved for future use.
1033h-1030h	TPM_INTERFACE_ID_1	Same as TPM_INTERFACE_ID_0
0038h-0034h	TPM_DATA_CSUM_ENABLE_1	Same as TPM_DATA_CSUM_ENABLE_0
003Bh-0038h	TPM_DATA_CSUM_1	Same as TPM_DATA_CSUM_0

Offset	Register Name	Description
107Fh-103Bh	Reserved	Reserved for future use.
1083h-1080h	TPM_XDATA_FIFO_1	Same as TPM_XDATA_FIFO_0
10BFh-1084h	Reserved	Reserved. These addresses are reserved by the chipset. A TPM should not respond to accesses to these addresses. Reserving this address range ensures that Software which issues writes larger than 1 byte to offset 0x0080 doesn't inadvertently encounter a register in a TPM in this space.
1EFFh-10C0h	Reserved	Reserved for future use.
1F03h-1F00h	TPM_DID_VID_1	Same as TPM_DID_VID_0
1F04h	TPM_RID_1	Same as TPM_RID_0
1F7Fh-1F05h	Reserved	Reserved for future use.
1F83h-1F80h	Reserved	Reserved for future use.
1F87h-1F84h	Reserved	Reserved for future use.
1F8Bh-1F88h	Reserved	Reserved for future use.
1F8Fh-1F8Ch	Reserved	Reserved for future use.
1FFFh-1F90h	Reserved	Vendor-defined configuration registers
Locality 2		
2000h	TPM_ACCESS_2	Used to gain ownership of a TPM for this locality.
2007h-2001h	Reserved	Reserved for future use.
200Bh-2008h	TPM_INT_ENABLE_2	Same as TPM_INT_ENABLE_0
200Ch	TPM_INT_VECTOR_2	Same as TPM_INT_VECTOR_0
200Fh-200Dh	Reserved	Reserved for future use.
2013h-2010h	TPM_INT_STATUS_2	Same as TPM_INT_STATUS_0
2017h-2014h	TPM_INTF_CAPABILITY_2	Same as TPM_INTF_CAPABILITY_0
201Bh-2018h	TPM_STS_2	Same as TPM_STS_0
2023h-201Ch	Reserved	Reserved for future use.
2027h-2024h	TPM_DATA_FIFO_2	Same as TPM_DATA_FIFO_0
202Fh-2028h	Reserved	Reserved for future use.
2033h-2030h	TPM_INTERFACE_ID_2	Same as TPM_INTERFACE_ID_0
0038h-0034h	TPM_DATA_CSUM_ENABLE_2	Same as TPM_DATA_CSUM_ENABLE_0
003Bh-0038h	TPM_DATA_CSUM_2	Same as TPM_DATA_CSUM_0
207Fh-203Bh	Reserved	Reserved for future use.
2083h-2080h	TPM_XDATA_FIFO_2	Same as TPM_XDATA_FIFO_0
20BFh-2084h	Reserved	Reserved. These addresses are reserved by the chipset. A TPM should not respond to accesses to these addresses. Reserving this address range ensures that Software which issues writes larger than 1 byte to offset 0x0080 doesn't inadvertently encounter a register in a TPM in this space.
2EFFh-20C0h	Reserved	Reserved for future use.
2F03h-2F00h	TPM_DID_VID_2	Same as TPM_DID_VID_0
2F04h	TPM_RID_2	Same as TPM_RID_0
2F7Fh-2F05h	Reserved	Reserved for future use.
2F83h-2F80h	Reserved	Reserved for future use.
2F87h-2F84h	Reserved	Reserved for future use.
2F8Bh-2F88h	Reserved	Reserved for future use.
2F8Fh-2F8Ch	Reserved	Reserved for future use.
2FFFh-2F90h	Reserved	Vendor-defined configuration registers
Locality 3		

Offset	Register Name	Description
3000h	TPM_ACCESS_3	Used to gain ownership of a TPM for this locality.
3007h-3001h	Reserved	Reserved for future use.
300Bh-30008h	TPM_INT_ENABLE_3	Same as TPM_INT_ENABLE_0
300Ch	TPM_INT_VECTOR_3	Same as TPM_INT_VECTOR_0
300Fh-300Dh	Reserved	Reserved for future use.
3013h-3010h	TPM_INT_STATUS_3	Same as TPM_INT_STATUS_0
3017h-3014h	TPM_INTF_CAPABILITY_3	Same as TPM_INTF_CAPABILITY_0
301Bh-3018h	TPM_STS_3	Same as TPM_STS_0
3023h-301Ch	Reserved	Reserved for future use.
3027h-3024h	TPM_DATA_FIFO_3	Same as TPM_DATA_FIFO_0
302Fh-3028h	Reserved	Reserved for future use.
3033h-3030h	TPM_INTERFACE_ID_3	Same as TPM_INTERFACE_ID_0
0038h-0034h	TPM_DATA_CSUM_ENABLE_3	Same as TPM_DATA_CSUM_ENABLE_0
003Bh-0038h	TPM_DATA_CSUM_3	Same as TPM_DATA_CSUM_0
307Fh-303Bh	Reserved	Reserved for future use.
3083h-3080h	TPM_XDATA_FIFO_3	Same as TPM_XDATA_FIFO_0
30BFh-3084h	Reserved	Reserved. These addresses are reserved by the chipset. A TPM should not respond to accesses to these addresses. Reserving this address range ensures that Software which issues writes larger than 1 byte to offset 0x0080 doesn't inadvertently encounter a register in a TPM in this space.
3EFFh-30C0h	Reserved	Reserved for future use.
3F03h-3F00h	TPM_DID_VID_3	Same as TPM_DID_VID_0
3F04h	TPM_RID_3	Same as TPM_RID_0
3F7Fh-3F05h	Reserved	Reserved for future use.
3F83h-3F80h	Reserved	Reserved for future use.
3F87h-3F84h	Reserved	Reserved for future use.
3F8Bh-3F88h	Reserved	Reserved for future use.
3F8Fh-3F8Ch	Reserved	Reserved for future use.
3FFFh-3F90h		Vendor-defined configuration registers
Locality 4		
4000h	TPM_ACCESS_4	Used to gain ownership of a TPM for this locality.
4007h-4001h	Reserved	Reserved for future use.
400Bh-4008h	TPM_INT_ENABLE_4	Same as TPM_INT_ENABLE_0
400Ch	TPM_INT_VECTOR_4	Same as TPM_INT_VECTOR_0
400Fh-400Dh	Reserved	Reserved for future use.
4013h-4010h	TPM_INT_STATUS_4	Same as TPM_INT_STATUS_0
4017h-4014h	TPM_INTF_CAPABILITY_4	Same as TPM_INTF_CAPABILITY_0
401Bh-4018h	TPM_STS_4	Same as TPM_STS_0
401Fh-401Ch	Reserved	Reserved for future use.
4023h-4020h	TPM_HASH_END	This signals the end of the hash operation. See Section 5.3 Locality-Controlled Functions for detailed description. This command SHALL be done on the LPC bus as a single write to 0x4020. Writes to addresses 0x4021 through 0x4023 are not decoded by a TPM.

Offset	Register Name	Description
4027h-4024h	TPM_HASH_DATA/ TPM_DATA_FIFO_4	Same as TPM_DATA_FIFO_0 except that this location is also used as the data port for the Locality 4 HASH procedure as defined in Section 5.3 Locality-Controlled Functions.
402Fh-4028h	TPM_HASH_START	This signals the start of the hash operation. See Section 5.3 Locality-Controlled Functions for detailed description. This command SHALL be done on the LPC bus as a single write to 0x4028. Writes to addresses 0x4029 through 0x402F are not decoded by TPM.
4033h-4030h	TPM_INTERFACE_ID_4	Same as TPM_INTERFACE_ID_0
0038h-0034h	TPM_DATA_CSUM_ENABLE_4	Same as TPM_DATA_CSUM_ENABLE_0
003Bh-0038h	TPM_DATA_CSUM_4	Same as TPM_DATA_CSUM_0
407Fh-403Bh	Reserved	Reserved for future use.
4083h-4080h	TPM_XDATA_FIFO_4	Same as TPM_XDATA_FIFO_0
40BFh-4084h	Reserved	Reserved. These addresses are reserved by the chipset. A TPM should not respond to accesses to these addresses. Reserving this address range ensures that Software which issues writes larger than 1 byte to offset 0x0080 doesn't inadvertently encounter a register in a TPM in this space.
4EFFh-40C0h	Reserved	Reserved for future use.
4F03h-4F00h	TPM_DID_VID_4	Same as TPM_DID_VID_0
4F04h	TPM_RID_4	Same as TPM_RID_0
4F7Fh-4F05h	Reserved	Reserved for future use.
4F83h-4F80h	Reserved	Reserved for future use.
4F87h-4F84h	Reserved	Reserved for future use.
4F8Bh-4F88h	Reserved	Reserved for future use.
4F8Fh-4F8Ch	Reserved	Reserved for future use.
4FFFh-4F90h		Vendor-defined configuration registers
Non-locality Specific Registers		
5FFFh-5000h	Reserved	Reserved for future use.
All addresses not defined in the table above		Reserved, reads return FFh; writes are dropped.

Start of informative comment

Subsequent sections provide implementation details on the defined registers. Note that registers which are aliased may have multiple versions, e.g., TPM_STS_x represents TPM_STS_0, TPM_STS_1, TPM_STS_2, TPM_STS_3, and TPM_STS_4.

End of informative comment

1. The DID/VID, RID, and all the TCG and vendor-specific registers MAY have only one physical copy.
 - a. If so implemented, these registers SHALL be accessible from any locality.
 - b. If implemented as separate physical registers, each copy SHALL hold the same data. See Section 7.1 FIFO Interface Locality Usage per Register and Table 47.
2. Handling Command FIFOs

Start of informative comment

Before issuing a command to a TPM, the Software reads a TPM_STS_x register to see whether a TPM's state allows it to accept commands.

Software sends commands to a TPM and reads results from a TPM using a data FIFO. When a TPM_STS_x.burstCount field is > 0, the data FIFO is ready to accept more data of a command (during a command's send phase) without incurring wait states, or return more data from a command (response from a command's completion). Since a TPM is not allowed to drop a cycle because of an internal stall, if a TPM cannot accept a write cycle or respond to a read cycle, it must insert wait states on the bus using the mechanism appropriate to the bus interface, i.e., stall the LPC bus using standard LPC wait syncs or insert one or more SPI wait cycles. See Section 7.4.5 Flow Control, for a detailed description of burstCount.

The FIFO is only a stack of bytes going into and out of a TPM. TPM_STS_x.burstCount indicates only the depth of the command FIFO, not the direction nor whether a TPM expects more data to be sent or received. TPM_STS_x.Expect and TPM_STS_x.DataAvail fields indicate to the Software when a TPM expects more data during a command's send phase or has more data to be read during a read results phase.

End of informative comment

3. A TPM SHALL maintain a TPM_STS_x register so that Software can determine whether a TPM is in a state where it can accept commands. A TPM SHALL NOT drop a cycle because of an internal stall. If a TPM cannot accept a write or read cycle, then a TPM SHALL stall the bus using the appropriate method for the bus (standard LPC Wait Syncs or SPI wait cycles).

6.5.2.2 Completion Command Details

6.5.2.2.1 Command Send

Start of informative comment

To send a command the Software must first set TPM_STS_x.commandReady = 1. Upon receipt of TPM_STS_x.commandReady, a TPM may set TPM_STS_x.Expect = 1 indicating it is ready to receive the command. When the data FIFO is ready to begin receiving the command data without incurring wait states, a TPM sets TPM_STS_x.burstCount > 0. A TPM uses TPM_STS_x.burstCount field to throttle the data into the data FIFO.

A TPM keeps TPM_STS_x.Expect = 1 until it receives all the expected data for the command. When a TPM receives all the data for the command (a TPM can calculate this using the command size parameter which is within the first 10 bytes of the command) it sets TPM_STS_x.Expect = 0 indicating to the Software that all data expected has been received.

The Software signals a TPM to begin executing the command by writing a 1 to a TPM_STS_x_tpmGo field. Upon receipt of TPM_STS_x_tpmGo = 1, a TPM begins executing the command.

End of informative comment

1. Upon receipt of TPM_STS_x.commandReady, a TPM SHALL prepare to receive a command.
2. When ready, a TPM SHALL set TPM_STS_x.commandReady = 1 and a TPM SHALL set TPM_STS_x.burstCount > 0 to indicate to Software that it can begin writing command data to the data FIFO without incurring wait states. A TPM MAY set TPM_STS_x.Expect = 1.
3. When a TPM receives the first Byte of the command data, it SHALL set TPM_STS_x.commandReady = 0 and TPM_STS_x.Expect = 1 as an indication to the Software that it expects further command data. A TPM SHALL use TPM_STS_x.burstCount to indicate to Software whether the data FIFO can accept more data without incurring wait states.
4. A TPM SHALL keep TPM_STS_x.Expect = 1 until it has received all of the data for this command. When a TPM reads the last byte of data from its data FIFO a TPM SHALL set TPM_STS_x.Expect = 0.
5. When TPM_STS_x.Expect = 0, a TPM SHALL ignore (drop) any additional data received in the data FIFO.
6. A TPM MAY ignore TPM_STS_x_tpmGo until TPM_STS_x.Expect is set to 0.

6.5.2.2.2 Data Availability

Start of informative comment

When a command completes, a TPM puts the results into the data FIFO, which is read via a TPM_DATA_FIFO_x register. Once a TPM has data that can be read, a TPM sets TPM_STS_x.dataAvail = 1 and it remains 1 until all data from the command response are read. After the last byte of the response is read, a TPM sets TPM_STS_x.dataAvail = 0. A TPM uses TPM_STS_x.burstCount field to throttle the response out of the data FIFO.

After sending a command, the Software reads a TPM_STS_x.dataAvail register to see if the response from a TPM is available, indicating a command has completed. If a TPM_STS_x.dataAvail field is 1, at least 1 byte of the command response data is available.

End of informative comment

1. Upon completing a command, a TPM SHALL place the command's response data in the data FIFO.
2. A TPM SHALL set TPM_STS_x.dataAvail = 1 as an indication to the Software that the command has completed and data is available to be read from the data FIFO. When the last byte of the response data is read from the data FIFO a TPM SHALL set TPM_STS_x.dataAvail = 0.

Note: A value of 1 indicates only that there is at least one byte in the data FIFO; it is not an indicator that the data can be read from the data FIFO. I.e., this is not the final indicator to Software that it can begin reading from the data FIFO. Software should also wait until TPM_STS_x.burstCount > 0, if avoiding wait states, see below.

3. A TPM SHALL set TPM_STS_x.burstCount > 0 to indicate to Software that it can begin reading the response data from the data FIFO without incurring wait states. Once the Software has started to read the response from the data FIFO, a TPM SHALL use TPM_STS_x.burstCount as an indicator to Software that data is available in the data FIFO.

6.5.2.3 Interface Specific Aborts

6.5.2.3.1 Command Aborts

Start of informative comment

There are several ways to cause a TPM to abort an executing command: TPM_ACCESS_x.Seize, TPM_STS_x.commandReady, and TPM_ACCESS_x.activeLocality. Because of implementation differences and the non-deterministic nature of some commands that may be executing, a TPM may not be able to respond to a command abort immediately or within a predictable time. This non-deterministic behavior causes driver design difficulties because the driver will not be able to distinguish between a TPM waiting normally and a TPM that has encountered an error and is not responsive. Therefore, a maximum amount of time is specified so TPM manufacturers have a design parameter that drivers can rely upon.

A TPM's internal state after an abort may be set to the state of a TPM prior to the aborted command or to the state it would have entered after completing the aborted command.

The purpose for a command abort when setting TPM_ACCESS_x.SEIZE or x.activeLocality is that a TPM cannot be allowed to "leak" information between localities. In other words, the response to a command sent from one locality cannot be returned to another locality.

Note: Because there is no requirement for a TPM to handle more than one operation at a time, there can be no actual and standardized TPM command to cause an abort. The method for signaling an abort to a TPM is by writing to specific registers.

End of informative comment

1. Upon a successful command abort, a TPM SHALL stop the currently executing command, clear the FIFOs, and transition to idle state.
2. The following operations SHALL cause a command abort:
 - a. Writing a 1 to TPM_STS_x.commandReady during the execution of a command.
 - b. Writing a 1 to TPM_STS_x.commandReady during the receipt of a command but before execution of a command.
 - c. Writing a 1 to TPM_ACCESS_x.Seize, but only when successful.

- d. Writing a 1 to TPM_ACCESS_x.activeLocality.
 - e. Successful completion of the HASH_START per Section 5.3, Locality Controlled Functions.
3. A TPM's internal state MAY either be in the pre-aborted command or post-aborted command state and SHALL not be in any intermediate state.
 4. For commands that do not cause RSA key generation, a TPM SHALL respond to an abort within TIMEOUT_A, see Section 6.5.1.4 Interface Timeouts. For commands that cause RSA key generation, a TPM SHALL respond to a request to abort the command within TIMEOUT_B, see Section 6.5.1.4 Interface Timeouts.

6.5.2.4 Access Register

Start of informative comment

The purpose of this register is to allow the processes operating at the various localities to share a TPM. The basic notion is that any locality can request access to a TPM by setting a TPM_ACCESS_x.requestUse field using its assigned TPM_ACCESS_x register address. If there is no currently set locality, a TPM sets current locality to the requesting one and allows operations only from that locality. If a TPM is currently at another locality, a TPM keeps the request pending until the currently executing locality frees a TPM. Software relinquishes a TPM's locality by writing a 1 to a TPM_ACCESS_x.activeLocality field. Upon release, a TPM honors the highest locality request pending. If there is no pending request, a TPM enters the "free" state.

There may be circumstances where the access to a TPM is "held" by either crashed or ill-behaved Software. For this reason, a TPM_ACCESS_x.Seize field may be used. It is generally assumed that Software executing at higher level localities is more trusted and less prone to crashing and better behaved at relinquishing a TPM. A TPM_ACCESS_x.Seize field allows higher-level localities to gain control of a TPM. This method, however, should be the exception rather than the common method for gaining access to a TPM.

In TPM 1.2, the relationship between the internal flag TPM_PERMANENT_FLAGS->tpmEstablished and the interface field TPM_ACCESS_x.tpmEstablishment is inverted logic. Therefore, when one is FALSE the other is TRUE. This is because Software accesses a TPM_PERMANENT_FLAGS->tpmEstablished field and expects "positive" logic, while hardware reads a TPM_ACCESS_x.tpmEstablishment and expects negative logic. To maintain some minimum level of backwards compatibility, the definition of the interface field TPM_ACCESS_x.tpmEstablishment maintains the same logic as in a TPM 1.2. TPM 2.0 does not have a flag that corresponds to a TPM_PERMANENT_FLAGS->tpmEstablished nor a command to reset a TPM_ACCESS_x.tpmEstablishment field, so an interface method has been added to this definition of the FIFO interface to provide the same functionality for a TPM 2.0

Software writing to a TPM_ACCESS_x register should set only one field to a 1 for each write. If a write to this register contains more than one field set to 1, the behavior of this register is undefined in this specification and the behavior between TPM implementations may differ.

Software needs to consider that there is no timeout condition defined for the time period between the release of one locality and when access to a subsequent locality is granted (i.e. TPM_ACCESS_x.activeLocality is set to 1), as this process happens practically immediately from a Software point of view.

Note: The HASH_START/_DATA/_END sequence is independent of the Access register. Software does not need to use the Access Register to send HASH_START/_DATA/_END, because those commands assert Locality 4. As a result, a TPM must always be ready to accept these commands when there is no active locality.

Note: A write to a TPM_ACCESS_x register with more than one field set to 1 may be ignored by a TPM.

End of informative comment

1. A TPM SHALL implement a TPM_ACCESS_x register, as documented in Table 27.
2. Any write operation to a TPM_ACCESS_x register, with more than one field set to a 1 MAY be treated as vendor specific.
3. For each write, fields containing a 0 SHALL be ignored.

Table 27 — Access Register

Abbreviation:		TPM_ACCESS_x		
General Description:		Used to gain ownership of a TPM		
Bit Descriptions:				
7	Read Only	tpmRegValidSts	Default: 0	This bit, when set to 1, indicates whether all other bits of this register contain valid values.
6	Read Only	Reserved	Default: 0	SHALL return 0
5	Read/Write	activeLocality	Default: 0	Read 0 = This locality is not active. Read 1 = This locality is active. Write 1 = Relinquish control of this locality
4	Read/Write	beenSeized	Default: 0	Read 0 = This locality operates normally or is not active Read 1 = Control of a TPM has been taken from this locality by another higher locality while this locality had its TPM_ACCESS_x.activeLocality bit set. Write 1 = Clear this bit.
3	Write Only	Seize	Reads always return 0	A write to this field forces a TPM to give control of a TPM to the locality setting this bit if it is the higher priority locality.
2	Read Only	pendingRequest	Default: 0	Read 1 = some other locality is requesting usage of a TPM Read 0 = no other locality is requesting use of a TPM
1	Read/Write	requestUse	Default: 0	Read 0 = This locality is either not requesting to use a TPM or is already the active locality Read 1 = This locality is requesting to use TPM and is not yet the active locality Write 1 = Request that this locality is granted the active locality
0	Read Only	tpmEstablishment	Default: 1	There are some special end cases (e.g., error conditions) where Software needs to know if a Dynamic OS has previously been established on this platform. This bit performs this function. The value of this flag SHALL be preserved across power and reset cycles. Read 0 = A Dynamic OS has been previously established on this platform Read 1 = A Dynamic OS has not been previously established on this platform

Field: *tpmRegValidSts*

Start of informative comment

If TPM_ACCESS_x.tpmRegValidSts is set, then all other fields [bits 0:6] of TPM_ACCESS_x are guaranteed to be correct.

If this field remains a 0 for longer than the period specified in Section 6.5.1.4 Timeouts, Software may assume that a TPM is broken and should not use it.

End of informative comment

1. For any read of a TPM_ACCESS_x. register, a TPM SHALL insert wait states (either an LPC Bus Long Wait Sync or SPI wait cycle) until the field TPM_ACCESS_x.tpmRegValidSts contains a valid logical level (i.e., 0 or 1) which represents its true state/value.

2. For all other register fields, which will contain a valid logical level only when `TPM_ACCESS_x.tpmRegValidSts = 1`, a TPM SHALL not return with `TPM_ACCESS_x.tpmRegValidSts = 1` in response to a read if any field contains an invalid logical level.
3. `TPM_ACCESS_x.tpmRegValidSts` SHALL be set to 1 within the Reset Timing requirements specified in Section 7.6 Reset Timing.

Field: Reserved

Start of informative comment

This field is reserved for future use.

End of informative comment

1. Writes to this field SHALL be ignored. A read from this field SHALL return 0.

Field: activeLocality

Start of informative comment

`TPM_ACCESS_x.activeLocality` has 3 functions:

It is used as an indicator to the Software to show whether the locality currently reading a `TPM_ACCESS_x` register is the active locality.

It is used by the Software that is currently accessing a TPM at the active locality to relinquish control of a TPM by writing a 1 to `TPM_ACCESS_x.activeLocality`

It can be used by the Software that has a currently pending request (to obtain the active locality) to cancel this pending request by writing a 1 to `TPM_ACCESS_x.activeLocality`.

The time from the request by any inactive locality until a TPM grants the request may vary depending on whether there is another locality to which a TPM has granted access.

After the request of a locality to become the active locality, `TPM_ACCESS_x.activeLocality` will be a 1 within `TIMEOUT_A` if there is no other locality active at this time, otherwise `TPM_ACCESS_x.activeLocality` will be a 1 only after the other locality has relinquished control of a TPM.

End of informative comment

Read:

1. If the requesting locality is the active locality, a TPM SHALL return `TPM_ACCESS_x.activeLocality = 1`.
2. If the requesting locality is not the active locality, a TPM SHALL return `TPM_ACCESS_x.activeLocality = 0`.

Write:

1. If a write occurs at the current active locality:
 - a. On a write of a 1 to the active locality's `TPM_ACCESS_x.activeLocality` field, a TPM SHALL:
 - i. Clear `TPM_ACCESS_x.activeLocality` field to 0 for the current locality.
 - ii. Relinquish control of a TPM for the current locality.
 - b. If there are pending requests from other localities, a TPM SHALL transfer control to the locality with the highest priority and set `TPM_ACCESS_x.activeLocality` to 1 for the new active locality.

Note: This locality becomes the new active locality.

2. If a write of 1 to `TPM_ACCESS_x.activeLocality` occurs at a locality which is not the current active locality and the locality performing the write has its `TPM_ACCESS_x.requestUse` set to 1 (e.g., there is a pending request for this locality which has not been granted), a TPM SHALL cancel the pending request from this locality.

3. If the requesting locality is not the active locality and its TPM_ACCESS_x.requestUse is 0, a write to this TPM_ACCESS_x.activeLocality SHALL be ignored.
4. TPM_ACCESS_x.activeLocality SHALL be set to 1 within TIMEOUT_A after TPM_ACCESS_x.requestUse has been set to 1 if a TPM is in the “free” state.
5. If there is another locality active at the time when a different locality sets its TPM_ACCESS_x.requestUse field to 1, this TPM_ACCESS_x.activeLocality SHALL be set to 1 within TIMEOUT_A after the original locality has relinquished control of its locality.

Start of informative comment

For the handling of changing locality during command execution and aborts see Section 6.5.1.1 Bus Aborts.

For example if Locality 2 is the current active locality and Locality 0 sets TPM_ACCESS_0.requestUse = 1, then Locality 0 has a pending request (i.e. TPM_ACCESS_0.requestUse is 1) and all other localities (1 to 4) have TPM_ACCESS_x.pendingRequest set to 1.

If now Locality 3 sets TPM_ACCESS_3.requestUse = 1 now Locality 3 also has a pending request with TPM_ACCESS_0.requestUse and TPM_ACCESS_3.requestUse being 1 and all other localities (0, 1, 2, 3 and 4) having TPM_ACCESS_x.pendingRequest set to 1.

Now Localities 0, 1, 2, 3, and 4 have TPM_ACCESS_x.pendingRequest set to 1 and localities 0 and 3 have their TPM_ACCESS_x.requestUse set to 1.

As soon as Locality 2 relinquishes control of a TPM by setting TPM_ACCESS_x.activeLocality to a 1, a TPM automatically transfers the control of a TPM to Locality 3 (because of the locality priority rules) and the pending request remains for Locality 0.

Now localities 1 to 4 have their TPM_ACCESS_x.pendingRequest set to 1 and Locality 0 has TPM_ACCESS_0.requestUse set to 1.

If now Locality 0 decides not to maintain the request to use a TPM, it sets TPM_ACCESS_0.activeLocality = 1 and consequently TPM_ACCESS_0.requestUse is cleared to 0 and TPM_ACCESS_x.pendingRequest of the localities 1 to 4 are cleared to 0 as well.

End of informative comment

Field: beenSeized

Start of informative comment

If a locality is the active locality, Software can use this field to determine whether the active locality has been taken away (i.e., seized) by another, higher priority locality and therefore the seized locality needs to abort an entire task and restart it after it has obtained the active locality again. This field can be cleared by the seized locality.

End of informative comment

1. TPM_ACCESS_x.beenSeized SHALL be set to a 1 when the active locality is seized.
2. A write of a 1 to TPM_ACCESS_x.beenSeized SHALL clear this field to a 0.

Field: Seize

Start of informative comment

The seize operation is a mechanism as a "last line of defense", if rogue Software does not relinquish control of a TPM and another, higher locality needs to obtain control of a TPM.

In this case, the Software of the higher locality (i.e. seizing locality) sets a TPM_ACCESS_x.Seize field to a 1 and then polls on TPM_ACCESS_x.activeLocality until this field returns a 1 (i.e. successful seize operation). At this point, the Software operating at the lower locality will be informed about the successful seize operation by TPM_ACCESS_x.beenSeized being set to a 1.

After the successful seize operation, the Software of the seizing locality reads a TPM_STS_x.commandReady field:

- 1) If a TPM_STS_x.commandReady field is a 0, Software should write a 1 to the field and then poll until it becomes a 1,
- 2) If a TPM_STS_x.commandReady field is 1 and TPM_STS_x.burstCount > 0, Software may immediately write the command to a TPM.

Seize operations from Locality 0 are ignored by a TPM, unless a TPM is in Locality None, since this operation has no real meaning for Locality 0.

For example, if Locality 0 is the currently active locality and Locality 1 writes a 1 to TPM_ACCESS_1.Seize, a TPM must clear a TPM_ACCESS_0.activeLocality field of locality 0, i.e. remove control of a TPM from Locality 0 and set TPM_ACCESS_x.beenSeized to 1. Consequently, a TPM must abort any currently executing command and stop accepting commands from Locality 0 as Locality 0 no longer has control of a TPM.

End of informative comment

1. If the write to TPM_ACCESS_x.Seize occurs from a locality of higher priority than the current locality:
 - a. A TPM SHALL clear a TPM_ACCESS_x.activeLocality fields for any active locality of lower priority than the locality seizing a TPM.
 - b. A TPM SHALL NOT change the state of a TPM_ACCESS_x.requestUse field for any locality except the one writing this field.
 - c. A TPM SHALL set TPM_ACCESS_x.activeLocality to a 1, clear a TPM_ACCESS_x.requestUse field to a 0 for the locality writing this field, and, if there are no other active requests, clear a TPM_ACCESS_x.pendingRequest field to 0 for all other localities.
 - d. A TPM SHALL abort any command that is currently in process, as defined in Section 6.5.2.3.1 Command Aborts.
2. If the write occurs from a locality that is equal to or lower than the current locality a TPM SHALL ignore the write.
3. A read from TPM_ACCESS_x.Seize SHALL return 0.

Field: pendingRequest

Start of informative comment

This field indicates whether a locality other than the currently active locality has requested to become the active locality. Software can use this field to determine whether it should relinquish control of a TPM so that the other locality can use it.

End of informative comment

1. When a locality writes a 1 to its TPM_ACCESS_x.requestUse, a TPM SHALL set TPM_ACCESS_x.pendingRequest to a 1 for all other localities.
2. If there are no pending requests, when the locality that has written a 1 to TPM_ACCESS_x.requestUse has obtained the control of a TPM as signified by TPM_ACCESS_x.activeLocality, TPM_ACCESS_x.pendingRequest for all other localities SHALL be cleared to 0.
3. When the locality with a pending request writes a 1 to its TPM_ACCESS_x.activeLocality field (i.e. cancels the pending request):
 - a. If there are no other pending requests, a TPM SHALL clear all TPM_ACCESS_x.pendingRequest fields to 0.
 - b. If there is one other pending request, a TPM SHALL clear a TPM_ACCESS_x.pendingRequest field to 0 for the locality with the active request.
 - c. If there are multiple pending requests, a TPM SHALL NOT clear any TPM_ACCESS_x.pendingRequest field to 0.
 - d. A TPM SHALL ignore writes to this field.

Field: requestUse

Start of informative comment

This field is used to request access to a TPM as the active locality. Software can write a 1 to this field when it needs to get control of a TPM. After the request is issued, Software must wait until its request to become the active locality is granted.

This field may only be cleared by writing a 1 to TPM_ACCESS_x.activeLocality for the requesting locality.

NOTE: Writing a zero to TPM_ACCESS_x.requestUse does not clear the pending request for this locality. See field: activeLocality for more information.

End of informative comment

1. When a locality writes a 1 to TPM_ACCESS_x.requestUse, a TPM SHALL set this field to 1 for the requesting locality.

Note: If a TPM_ACCESS_x.requestUse is already set to 1, a TPM SHALL ignore writes of 1 to this field.

2. When the locality that has set its TPM_ACCESS_x.requestUse has been granted control of a TPM as signified by TPM_ACCESS_x.activeLocality set to 1, a TPM SHALL clear a TPM_ACCESS_x.requestUse field to 0 for the requesting locality.
3. When a locality cancels a pending request, signified by writing a 1 to TPM_ACCESS_x.activeLocality, a TPM SHALL clear this field to 0 for the requesting locality.
4. A TPM SHALL ignore writes of 0 to TPM_ACCESS_x.requestUse.

Field: tpmEstablishment

Start of informative comment

TPM_ACCESS_x.tpmEstablishment is a register field which indicates whether a Dynamic OS has been launched. The reason this register field uses inverted logic is to allow systems without TPMs to indicate, using this register, that no Dynamical OS has been launched. Since the default state of the register field is 1, Reading from a device that doesn't exist (there is no device to claim the read request) returns a value of all ones. Therefore, a read access to a TPM's access register when there is no TPM present returns as if no Dynamic OS has been established.

End of informative comment

1. If a TPM_ACCESS_x.tpmRegValidSts is set to 1, any read of a TPM_ACCESS_x.tpmEstablishment field SHALL reflect the correct value.
2. If TPM_ACCESS_x.tpmRegValidSts is cleared to 0 (i.e., TPM_ACCESS register is not valid):
 - a. TPM_ACCESS_x.tpmEstablishment MAY be a 0.
 - b. TPM_ACCESS_x.tpmEstablishment SHALL NOT be a 1 unless that is the correct value of the field.
3. TPM_ACCESS_x.tpmEstablishment SHALL be 0 prior to initialization and update of the D-RTM PCR at the completion of HASH_END.
 - a. If TPM_ACCESS_x.tpmEstablishment is 0:
 - i. TPM_ACCESS_x.tpmEstablishment SHALL remain 0 until a TPM receives a write of 1 to TPM_STS_x.resetEstablishmentBit
 - ii. The value of TPM_ACCESS_x.tpmEstablishment SHALL NOT change across power or reset cycles.
4. TPM_ACCESS_x.tpmEstablishment SHALL be 1 if no D-RTM HASH_END has been executed.
5. If TPM_ACCESS_x.tpmEstablishment is 1, receipt of a D-RTM HASH_END command SHALL clear it to 0 within TIMEOUT_A.
6. TPM_ACCESS_x.tpmEstablishment SHALL be duplicated across Localities 0-4.

6.5.2.5 Status Register

Start of informative comment

A TPM_STS_x.commandReady field is functionally overloaded. If there is no command being executed, a write to this field is an indicator to a TPM that it must prepare to receive a command. If there is a command being executed, a write to this field serves as an abort of that command. If a command has completed and the results have been read, a write to this field allows a TPM to free internal resources (including the Read and Write FIFOs) and proceed with background or other processes allowed during idle time. A TPM may be designed in a manner that allows the first write to this field to clear and free a TPM's resources and make it ready to receive a command.

Software must be prepared to send two writes of a 1 to this field: the first to indicate successful read of all the data, thus clearing the data from the Read FIFO, and freeing a TPM's resources, and the second to indicate to a TPM it is about to send a new command. The time between receiving the data from a command and sending the first write to this field should be very short to allow TPMs that perform background processing to proceed. The time between the first write and the second indicates the beginning of a new command is arbitrary.

Software may be written such that the second write to this field is only necessary if a TPM does not respond with a ready after the first write. In this case, the Software, after writing a 1 to this field indicating the receipt of the data, may query this field. If a TPM sets this field to a 1 indicating its readiness to receive a command, the Software may proceed to send the command without writing a 1 to this field.

End of informative comment

6.5.2.5.1 TPM Status Register States

For each write to this register, there SHALL be only one field set to a 1. If a TPM receives a write with more than one field set, a TPM SHALL ignore the entire cycle. For each write, fields containing 0 are ignored.

A TPM is in one of the following defined states:

1. *Command Reception* occurs between the write of the first byte of a command to the WriteFIFO following a ready state and the receipt of a write of 1 to TPM_STS_x.tpmGo.
2. *Command Execution* occurs after receipt of a 1 to TPM_STS_x.tpmGo and a TPM setting TPM_STS_x.commandReady:dataAvail to a 1, unless the command is aborted as defined in Section 6.5.2.3.1 Command Aborts.
3. *Command Completion* occurs after completion of a command (indicated by a TPM setting a TPM_STS_x.commandReady:dataAvail to a 1) and before a write of a 1 by the Software to TPM_STS_x.commandReady.
4. *Idle* is any time after Command Completion followed by the write of a 1 by the Software to TPM_STS_x.commandReady, following locality change, or a command abort. Idle is the initial state of TPM upon completion of _TPM_INIT.
5. *Ready* is any time a TPM is ready to receive a command, as indicated by TPM_STS_x.commandReady being set.

Start of informative comment

The following informative diagram is derived from the above normative statements. It is informative and only for illustrating diagrammatically the above TPM states and their transitions. The numbers in parentheses reference the states represented by row numbers in Table 31.

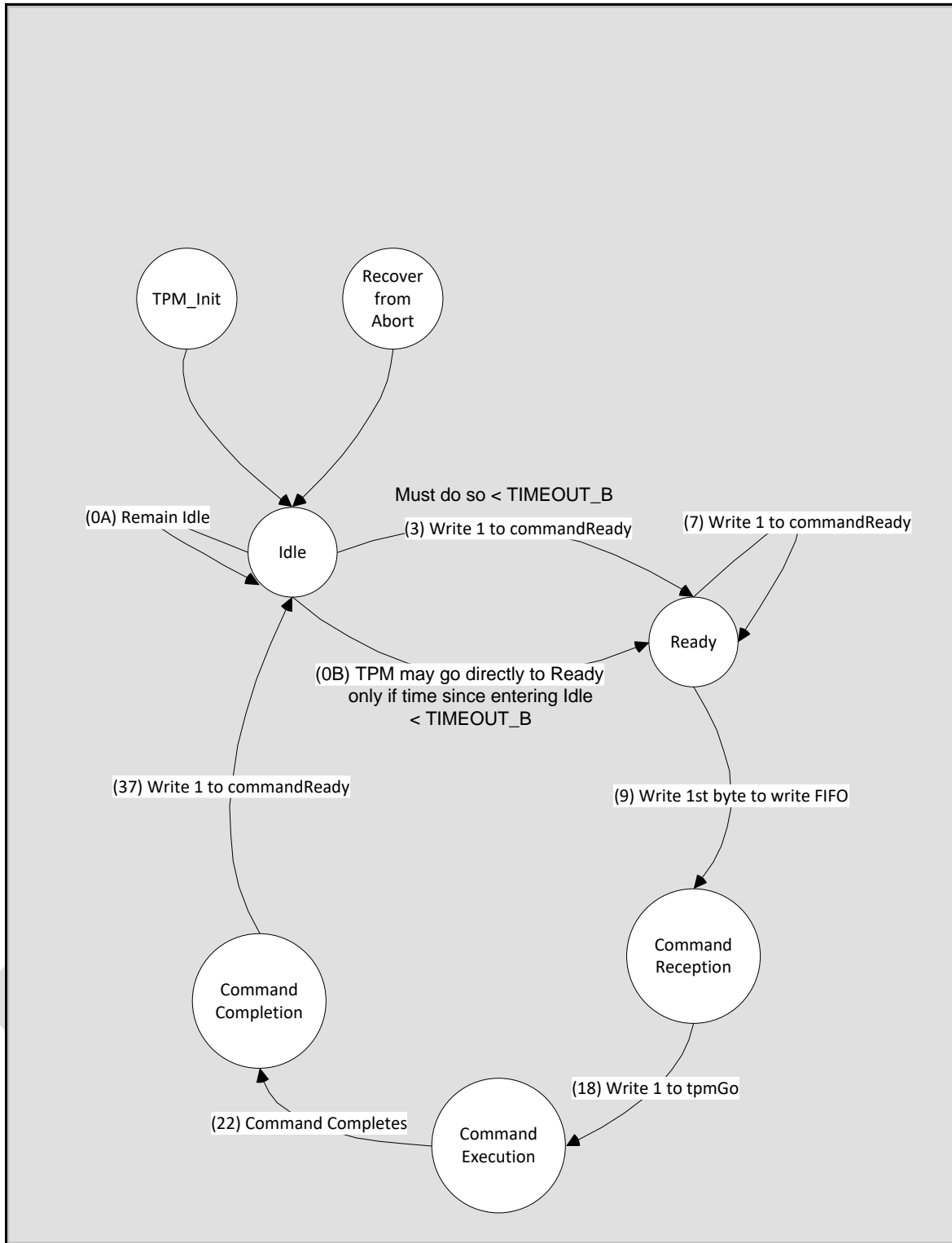


Figure 3 — State Transition Diagram

End of informative comment

6.5.2.5.2 Bus Access of the Status Register

- For any read from a TPM_STS_x register, a TPM SHALL assert zero or more wait states (either an LPC Bus Long Wait Sync or SPI wait cycles) until all fields in the register, except TPM_STS_x.dataAvail and TPM_STS_x.Expect, contain a valid logical level (i.e., 0 or 1) which represents their true state/value.
- The register fields TPM_STS_x.dataAvail and TPM_STS_x.Expect will contain a valid logical level only when TPM_STS_x.stsValid=1.
- A read of TPM_STS_x.stsValid SHALL not return 1, if TPM_STS_x.dataAvail (while reading results) or TPM_STS_x.Expect (while sending a command) contains an invalid logical level, respectively.
- A TPM SHALL implement the Status Register as documented in Section 6.5.2.5 Status Register.

Table 28 — Status Register

Abbreviation:			TPM_STS_x	
General Description:			Contains general status details	
Bit Descriptions:				
31:28	Read Only	reserved	Reads always return 0	
27:26	Read Only	tpmFamily		TPM Family Identifier 00: TPM 1.2 Family 01: TPM 2.0 Family 10: Reserved future use 11: Reserved for future use
25	Write Only	resetEstablishmentBit	Reads always return 0 Write of 1 resets the Established Flag	Reads always return 0 Writes (0): Ignored Writes (1): Reset TPM_ACCESS_x.tpmEstablished bit if the write occurs from Locality 3 or 4. Valid indicator: NA
24	Write Only	commandCancel	Write Only	Reads always return 0 A write of a 1 to this field after tpmGo and before dataAvail aborts the currently executing command, resulting in a response of TPM_RC_CANCELLED. A write of 1 to this field after dataAvail and before tpmGo is ignored by a TPM. Writes of 0 may be ignored. Valid indicator: NA
23:8	Read Only	burstCount	Default = number of consecutive writes that can be done to a TPM	Indicates the number of bytes that a TPM can return on reads or accept on writes without inserting wait states on the bus. Valid indicator: NA
7	Read Only	stsValid	Default: 0	This field indicates that TPM_STS_x.dataAvail and TPM_STS_x.Expect are valid. Read of 1 indicates that TPM_STS_x.davaAvail and TPM_STS_x.Expect contain a valid value. Valid indicator: N/A
6	Read/Write	commandReady	Default: 0	Read of 1 indicates that a TPM is ready. Write of 1 causes a TPM to transition its state. Valid indicator: N/A

Abbreviation:			TPM_STS_x	
General Description:			Contains general status details	
Bit Descriptions:				
5	Write Only	tpmGo	Reads return 0	always After Software has written a command to a TPM and sees that it was received correctly, Software SHALL write a 1 to this field to cause a TPM to execute that command. Valid indicator: N/A
4	Read Only	dataAvail	Default: 0	This field indicates that a TPM has data available as a response. When set to 1, Software MAY read the ReadFIFO. A TPM SHALL clear the field to 0 when it has returned all the data for the response. Valid indicator: TPM_STS_x.stsValid = 1
3	Read Only	Expect	Default: 0	A TPM sets this field to a value of 1 when it expects another byte of data for a command. It clears this field to a value of 0 when it has received all the data it expects for that command, based on a TPM size field within the packet. Valid indicator: TPM_STS_x.stsValid = 1
2	Read Only	selfTestDone	Default: 0	This field indicates that a TPM has completed all self-test actions following a TPM2_SelfTest (FULLTEST = No) command. Read of 0 indicates self-test is not complete. Read of 1 indicates self-test is complete.
1	Write Only	responseRetry	Reads return 0	always Software writes a 1 to this field to force a TPM to re-send the response. Reads SHALL return 0. Valid indicator: N/A
0	Read Only	Reserved (0)	Reads return 0	always

Field: *resetEstablishmentBit*

Start of informative comment

This field is used to replace functionality provided by the TSC_ResetEstablishmentBit command in TPM 1.2. This command is not present in TPM 2.0, but this interface mechanism provides equivalent functionality.

This field is used to reset the state of a TPM_ACCESS_x.tpmEstablished field, once that bit has been set to 0 by a D-RTM sequence. A write to this field will be processed by a TPM as if a command has been received. Once this field is written, a TPM will clear commandReady until a TPM_ACCESS_x.tpmEstablished flag has been cleared.

This field can only be written from Localities 3 and 4.

A TPM will treat a write to this bit as if a TPM has received a Command, and will set TPM_STS_x.dataAvail to 1 once the actions of the operation have been completed. As a TPM may or may not place data in the FIFO, Software should ignore this data. It will be overwritten in the next command cycle.

End of informative comment

1. Reads to TPM_STS_x.resetEstablishmentBit SHALL always return 0.
2. For Localities 0-2, writes are ignored.
3. For Localities 3 and 4
 - a. Writes of 0 are ignored

- b. When in the Ready or Idle state, writes of 1 set TPM_ACCESS_x.tpmEstablished bit to a 1 and clear TPM_STS_x.resetEstablishmentBit bit to 0 within TIMEOUT_A.

Field: commandCancel

Start of informative comment

Cancel may be used by Software to request a TPM to terminate processing the current command. Software might request this because the system is going to a lower power state. A TPM will either complete the currently processing command and return the result or will cancel the command and return the return code TPM_RC_Canceled. In general, for long running commands, a TPM may have checkpoints in its code to check the state of the Cancel field. If, at one of these checkpoints, a TPM sees a Command Cancel request, a TPM has the option of canceling the command or completing the command. TPMs are not required to perform this check. TPMs that can finish all commands within the required timeouts may return the command response instead of cancelling the command.

Cancel is an asynchronous input. Driver writers should take care in use of this function in their drivers, as a TPM could enter a state where it would process no commands. This case may occur if Software sends a Cancel but fails to clear it. A TPM may be implemented such that it will continue to cancel commands or will complete the command and transition to Command Completion with a result. Alternatively, TPMs may be implemented so that they only cancel a single command. Driver writers should take care to send command Cancel requests only when a TPM is in Command Execution state and to only clear the field when a TPM has exited Command Execution to avoid unexpected behavior.

In the event that Software cancels a command by writing a 1 to TPM_STS_x.commandCancel, and then writes a 0 to TPM_STS_x.commandCancel, a TPM's behavior is not defined by this specification. Depending on the timing of both write operations a TPM might not see the first write. If it does see the first write, it is not required to cancel the command if it is easier to finish the command processing, and thus a write to 0 to "cancel the Cancel" would have no effect.

End of informative comment

1. When a TPM is in the Command Execution state and TPM_STS_x.commandCancel is set to 1:
 - a. A TPM MAY terminate command processing,
 - b. A TPM SHALL return a response and transition to the Command Completion state:
 - i. If the command is successfully completed, a TPM SHALL return the command response.
 - ii. If the command is terminated, a TPM SHALL return TPM_RC_CANCELED.
 - iii. A TPM SHALL complete the command or cancel it within TIMEOUT_B.
2. When a TPM is in any state other than Command Execution, a TPM SHOULD ignore TPM_STS_x.commandCancel.
3. A TPM MAY ignore writes of 0.

Field: BurstCount:

Start of informative

It's helpful to understand burstCount by first explaining it in the context of an example implementation. For example, a TPM's firmware processes commands once they are received by a TPM. Software, however, does not directly interact with a TPM's firmware. Rather, it sends and receives data via hardware registers called a data FIFO. During a command send phase, Software writes command data directly to the data FIFO which the firmware reads from the FIFO. There is no relationship between the size or amount of data sent to the data FIFO (from either side) and the size of the command or the command's response. The data FIFO, therefore, can be thought of as a hardware buffer between the Software and a TPM's firmware. The value in the burstCount field is simply the number of bytes that can be written or read from the data FIFO (i.e., the hardware buffer) at any one time without incurring wait states. It will likely require multiple writes (for command send) or multiple reads (for response reads) to and from the data FIFO to send a command and read a response.

It is expected that the data FIFO is sufficiently fast so that, provided there is room (during command send) and bytes available (during a read response) in the data FIFO, all writes and reads will occur without any wait states. Therefore,

burstCount is defined as the number of bytes that can be written to or read from the data FIFO by the Software without incurring a wait state.

If Software is written to avoid incurring wait states, Software should read this field and write or read the number of bytes indicated. Once that number of bytes has been written to a TPM, a TPM may set burstCount = 0. If Software is using burstCount to avoid incurring wait states, Software should wait while burstCount is 0, checking for a non-zero value by polling on burstCount until it is greater than 0. Once burstCount is greater than 0, Software may resume writing or reading data up to the new burstCount value without incurring wait states.

Again, there is no relationship between the size of the data FIFO and the value in the burstCount field, versus the size of the command or response data. On a command send, Software must not pad to the data FIFO, nor must it read more response data than indicated by the command's response size value. For example, if after sending several data FIFO's worth of command data to a TPM, there are seven bytes left to send, even if burstCount = 16, Software must still only send seven bytes. Conversely, on response read if there are only three bytes left of the response to read, Software must only read three bytes from the data FIFO even if burstCount = 16.

This field may be dynamic or static as indicated by TPM_INTF_CAPABILITY_x.burstCountStatic.

A dynamic burstCount field reports a changing number of bytes that can be read or written. For example, on command send, as data is written to the data FIFO the burstCount field is decremented by the number of bytes written indicating there are fewer bytes available to write into the data FIFO. As the firmware reads the data out of the data FIFO the burstCount field is incremented indicating there are more bytes available in the FIFO. Conversely, on response read, as Software reads data from the data FIFO the burstCount field decrements indicating there are fewer bytes in the data FIFO available to read without incurring a wait state. As firmware writes response data to the data FIFO the burstCount field increments indicating there are more bytes available to read without incurring a wait state.

A static burstCount field reports a fixed number of bytes that can be read or written. Software reads burstCount and must keep track of that value. Once Software begins to write, for command send, or read, for response read, a TPM sets burstCount = 0 until the fixed value is written or read. Only after the fixed number of bytes have been written or read will the burstCount field contain a nonzero value for Software to read. Note that in this case, burstCount is a fixed value from _TPM_INIT, allowing Software to read burstCount once when the Software is first initialized and to save the value, and to use the saved value until the next _TPM_INIT. In this case after the initial read of burstCount, Software only needs to look at whether burstCount is a zero or non-zero value.

End of informative

1. For dynamic burstCount (i.e., TPM_INTF_CAPABILITY_x.burstCountStatic == 0):
 - a. For command send phase
 - i. If the data FIFO is not ready to receive data from the Software without incurring wait states, a TPM SHALL set burstCount = 0.

Note: this rule applies not just to the beginning of the command send phase but at any time during the command send phase. E.g., after several writes to the data FIFO have occurred, the Software may fill the data FIFO completely because the firmware cannot read and process the data as fast as Software can write it.

- ii. When a TPM is ready to receive data in the data FIFO, a TPM SHALL set burstCount equal to the number of bytes Software can write without incurring wait states.
 1. As data is received from Software into the data FIFO, a TPM SHALL decrement burstCount.
 2. As a TPM (e.g., firmware) reads data from the data FIFO a TPM SHALL increment burstCount.
 - b. For response read phase
 - i. If the data FIFO is not ready to return data to the Software without incurring wait states, a TPM SHALL set burstCount = 0.

Note: this rule applies not just to the beginning of the response read phase but at any time during the response read phase. E.g., after several reads from the data FIFO have occurred the Software may read all the data in the data

FIFO (i.e., empty the data FIFO) because the firmware cannot place response data in the FIFO as fast as Software can read it.

- ii. When a TPM is ready for Software to read data from the data FIFO, a TPM SHALL set burstCount equal to the number of bytes Software can read without incurring wait states.
 - 1. As Software reads data from the data FIFO, a TPM SHALL decrement burstCount.
 - 2. As a TPM (e.g., firmware) writes data to the data FIFO a TPM SHALL increment burstCount.
- 2. For static burstCount (i.e., TPM_INTF_CAPABILITY_x.burstCountStatic == 1):
 - a. For command send phase
 - i. If the data FIFO is not ready to receive data from the Software without incurring wait states, a TPM SHALL set burstCount = 0.
 - ii. When the data FIFO is ready to receive data from the Software without incurring an LPC wait state, a TPM SHALL set burstCount equal to the maximum number of bytes that can be transferred by Software to a TPM without incurring wait states.
 - iii. Upon receipt of the first command data byte a TPM SHALL set burstCount = 0. The burstCount field SHALL remain set = 0 until the indicated number of bytes have been sent by Software to the data FIFO.
 - iv. Once a TPM has received the indicated number of bytes in the data FIFO, a TPM must set burstCount equal to the value defined in normative 2.a.i or 2.a.ii.

Note: for static burstCount, burstCount is a fixed value and SHALL NOT change after _TPM_INIT until the next _TPM_INIT.

- b. For response read phase
 - i. If the data FIFO is not ready to return data to the Software without incurring wait states, a TPM SHALL set burstCount = 0.
 - ii. When the data FIFO is ready for Software to read data without incurring wait states, a TPM SHALL set burstCount equal to the maximum number of bytes that can be read by Software without incurring LPC wait sync or SPI wait cycles.
 - iii. Upon Software's read of the first response data byte, a TPM SHALL set burstCount = 0. The burstCount field SHALL remain = 0 until Software has read the indicated number of bytes from the data FIFO.
 - iv. Once the Software has read the indicated number of bytes from the data FIFO, a TPM SHALL set burstCount equal to the value defined in normative 2.b.i or 2.b.ii.

Note: for static burstCount, burstCount is a fixed value and SHALL NOT change after _TPM_INIT until the next _TPM_INIT.

- 3. Following a write to TPM_STS_x.responseRetry or a command abort operation, any value previously read from a TPM_STS_x.burstCount field is invalid until TPM_STS_x.DataAvail = 1.
- 4. Timeout:
 - a. For command send: After TPM_STS_x.commandReady is set to 1, TPM_STS_x.burstCount SHALL be non-zero within the time specified by TIMEOUT_A, see Section 6.5.1.4 Interface Timeouts.
 - b. For response read: After TPM_STS_x.DataAvail == 1, TPM_STS_x.burstCount SHALL be non-zero within the time specified by TIMEOUT_A, see Section 6.5.1.4 Interface Timeouts.
- 5. A TPM SHALL report the correct count even though there is a time delay in returning the 2 bytes on the LPC bus.

Start of informative

There are many ways to ensure the correct count is always reported. A few examples are below:

Return 0x00 for the upper byte in all cases. This limits the field to 255 bytes, but that is a sufficiently large number that polling on this register every 255 bytes is insignificant in terms of performance.

Guarantee that the count does not change between the read of the first byte and the read of the second byte. This could be done if the hardware updates the count field at the time of the previous read or write and does not change it until the next read or write. Alternatively, it could be done if hardware latches both bytes of the count that is returned on the read of the first byte and returns the latched second byte on the next read. In this case, if there is a read or write of the data before the next read of a TPM_STS_x.burstCount, then the latch is reset.

Use static burstCount:

Note: that there could be the case where internally a TPM is processing the FIFO and TPM_STS_x.burstCount is dynamic, whereby the count is changing even though there are bus transactions. In this case, the read of the low byte might not be synchronized with the high byte – hardware must not allow this condition.

End of informative

Field: stsValid

Start of informative comment

TPM_STS_x.stsValid gates reads to the fields TPM_STS_x.dataAvail and TPM_STS_x.Expect. If TPM_STS_x.stsValid is not set, then TPM_STS_x.dataAvail and TPM_STS_x.Expect are not guaranteed to be correct. If a TPM does not support the stsValid Interrupt, Software that is using TPM_STS_x.dataAvail or TPM_STS_x.Expect must poll on TPM_STS_x register until TPM_STS_x.stsValid is set. Software should not use the contents of the status register if this field is 0.

End of informative comment

1. A TPM SHALL set a TPM_STS_x.stsValid field within TIMEOUT_A, see Section 6.5.1.4 Interface Timeouts after the last data cycle to this register is received.
2. A TPM SHALL not set a TPM_STS_x.stsValid field to 1 unless either a TPM_STS_x.Expect field is valid in the command Completion state or TPM_STS_x.dataAvail field is valid in the command Reception state.

Field: commandReady

Start of informative comment

TPM_STS_x.commandReady is a dual-function field. It is used by a TPM to indicate readiness to receive a command. Software uses this field to initiate a command sequence with a TPM.

Note: Software should be designed such that it checks this field before sending any new data to a TPM data FIFO. This allows Software to know a TPM's state. Software should never send data to a TPM when this field is set to 0, indicating a TPM is not ready. After Software has successfully received data from a TPM, it should write a 1 to this field to signal to a TPM that the response was correctly received.

End of informative comment

Read:

1. A TPM is in the *Ready* state when this field is set to 1.
 - a. A TPM SHALL not enter the *Ready* state unless both the ReadFIFO and WriteFIFO are empty.
 - b. A TPM SHALL clear this field to 0 when the first byte of data is received by the WriteFIFO.
2. A TPM is not in a *Ready* state when this field is set to 0.

Write:

1. A write of 0 to this field SHALL be ignored.
2. Upon a write of a 1 to this field.
 - a. When in the *Ready* state, a TPM SHALL ignore this write and remain in the *Ready* state.

- b. When in the *Idle* state, a TPM SHALL enter the *Ready* state within the time TIMEOUT_B, see Section 6.5.1.4 Interface Timeouts.
- c. When in the *Command Reception* state, a TPM SHALL treat this as an abort according to Section 6.5.2.3 Command Aborts.
- d. When in the *Command Completion* state:
 - i. A TPM SHALL clear the ReadFIFO and the WriteFIFO.
 - ii. A TPM SHALL enter either the *Idle* state or the *Ready* state.
 - iii. If a TPM does not enter the *Ready* state within time TIMEOUT_B, see Section 6.5.1.4 Interface Timeouts, it SHALL enter the *Idle* state.
- e. When in the *Command Execution* state, a TPM SHALL cause the currently executing command to be aborted according to Section 6.5.2.3 Command Aborts.

Field: tpmGo

Start of informative comment

This field is used by Software to tell a TPM to execute the received command. Execution of the command may take from several seconds to minutes for certain commands, such as key generation. Software should confirm a TPM has received the complete command by reading a TPM_STS_x.stsValid and TPM_STS_x.Expect fields. This field is write-only.

End of informative comment

1. A TPM SHALL execute the received command on a write of 1 to this field.
2. A TPM SHALL ignore a write of 0 to this field and SHALL NOT return an error.
3. A TPM SHALL return 0 on a read request.

Field: dataAvail

Start of informative comment

A TPM sets this field when it is ready to return the response. The validity of this field is determined by TPM_STS_x.stsValid, as defined in normative text for the *Field: stsValid*.

To detect overruns/under runs, Software SHOULD read TPM_STS_x.dataAvail before it reads what it thinks is the last byte of the response. If TPM_STS_x.dataAvail is 1, then there is at least 1 more byte to read. Software reads the last byte and re-reads TPM_STS_x.dataAvail. In this case, TPM_STS_x.dataAvail should be 0; since, if things are working correctly, there is no more data to return. If TPM_STS_x.dataAvail is still 1, then a TPM has more data to return, and Software is out of sync with the hardware. Therefore, Software should set TPM_STS_x.responseRetry to force a TPM to resend the response.

If a read of TPM_STS_x.dataAvail returns a 0 before Software reads the last byte, a TPM thinks it has no more data to return, while Software still expects 1 more byte. In this case, Software SHALL set TPM_STS_x.responseRetry to force a TPM to resend the response.

If the DataAvailInt interrupt is not supported, Software must poll on a TPM_STS_x register until TPM_STS_x.dataAvail is set.

End of informative comment

1. A TPM SHALL not set dataAvail to a 1 unless the TPM has completed command execution and data is ready to be read.
 - a. A TPM SHALL set this field when data is present in the ReadFIFO.
 - b. A TPM SHALL set this field when Software writes to TPM_STS_x.responseRetry.
 - c. A TPM SHALL make data available in the ReadFIFO when this field is set.
 - d. A TPM SHALL set the DataAvailInt interrupt, if the interrupt is supported, after setting this field.
2. A TPM SHALL set this field to zero if the TPM is either not ready to transmit data or has returned the last byte of data.

- a. A TPM SHALL clear this field to 0 when the ReadFIFO is empty because either the TPM has sent all the data or is not ready to send data.
 - b. A TPM SHALL clear this field to 0 when Software sets TPM_STS_x.commandReady = 1 even though the response data has not been read.
3. This field SHALL be valid if TPM_STS_x.stsValid is set.

Field: Expect

Start of informative comment

This field is set by a TPM when it expects to receive data from Software. The validity of this field is determined by TPM_STS_x.stsValid.

A TPM will set this field once it starts receiving data. The field will stay set until a TPM receives at least 10 bytes, as this is the smallest valid command length. A TPM will use the first 10 bytes to determine the actual length of the command. If the length is longer than 10 bytes. This field will stay set until a TPM receives the full command.

End of informative comment

1. Software should examine the Expect field before and after sending the last byte to ensure that a TPM has received the full command. If the Expect field is set to 1 before Software sends the last byte of the command, there is no error condition. Software should send the last byte and check Expect again. If the Expect field is set to a 0, the command has been successfully received. If the value of the Expect field is 0 prior to Software sending the last byte or is 1 after Software sends the last byte, an error has occurred, and Software should restart the command.
2. A TPM SHALL set this field to 1 when in the command Reception state and MAY set this field to 1 when in the Ready state.
3. A TPM SHALL NOT set this field to 1 in any other state.
4. A TPM SHALL clear this field to 0 when in any other state.
5. This field SHALL be valid if TPM_STS_x.stsValid is set.

Field: selfTestDone

Start of informative comment

This field is set by a TPM to indicate the status of a TPM's self-test following receipt of a TPM_SelfTest command. The field will be 0 if a TPM has capabilities remaining to be tested. Once a TPM completes its self-test, it sets this bit to a 1. This field is always valid.

End of informative comment

1. A TPM SHALL set this field to 1 when all of the actions required to complete the command TPM_SelfTest are done.
2. A TPM SHALL NOT set this field to 1 unless it has completed all the actions required by TPM_SelfTest.
3. A TPM SHALL ignore writes to this field and SHALL NOT return an error.

Field: responseRetry

Start of informative comment

This field is set by Software to force a TPM to resend a response without sending a command. This may occur if a TPM exceeds the timeout defined for the response. Software should implement a retry counter and set this field if the retry counter is less than its threshold. This field is write-only.

End of informative comment

1. A TPM SHALL resend the last response on a write of 1 to this field.
2. A TPM SHALL ignore writes of 0 to this field and SHALL NOT return an error.

- A TPM SHALL return 0 on a read request.

6.5.2.6 Data FIFO Register

Start of informative comment

This register is the port used by a TPM to return data and status to Software. Return packets for commands are multiple bytes but are read by Software in 1-byte increments. Software should read a TPM_STS_x.burstCount field to determine how many consecutive bytes it can read. Software should read a TPM_STS_x.burstCount field for better general system performance.

As the HASH_START/_DATA/_END interface commands are independent of a TPM_ACCESS_x register, processes calling these commands should not poll TPM_STS_x.burstCount and should send data to a TPM using only the interface protocols and bus speeds.

End of informative comment

Table 29 — Data FIFO Register

Abbreviation:		TPM_DATA_FIFO_x	
General Description:		Data port for TPM	
Bit Descriptions:			
7:0	Read/Write	Default: undefined	Reads to this register return Command Response data. Writes to this register contain Command Send Data

- When TPM_STS_x.stsValid is set to 1 and TPM_STS_x.dataAvail is cleared to 0, a TPM SHOULD return FFh to any read request to the Data FIFO register.
- A TPM SHALL NOT drop a write on the bus when a TPM is not able to accept it. Instead, it SHALL insert one or more wait states, according to the interface protocol (e.g. wait-sync the LPC bus or an SPI wait cycle according to Section 7.4.5 Flow Control, or I2C clock stretching as defined in Section 8.1.4).

6.5.2.7 Interface Capability

Start of informative comment

This register is valid only for a FIFO interface. Bits 9 and 10 allow a TPM to communicate the maximum data transfer size it supports. A TPM may support larger transactions on LPC.

End of informative comment

- This register SHALL be valid if TPM_INTERFACE_ID_x.InterfaceType or TPM_CRB_INTF_ID_x.InterfaceType is set to 0000 or 1111. For all other values of TPM_INTERFACE_ID_x.InterfaceType or TPM_CRB_INTF_ID_x.InterfaceType, this register is invalid.
- The DataTransferSizeSupport field indicates the maximum transfer size supported by a TPM.
- If a TPM supports only legacy transfer sizes, the DataTransferSizeSupport field SHALL be set to '00'.
 - Reads and Writes SHALL only be accepted at the offset for a TPM_DATA_FIFO.
 - TPM SHALL perform a bus abort on transactions to a TPM_XDATA_FIFO.
- If a TPM supports any DataTransferSizeSupport value other than '00':
 - A TPM SHALL support a TPM_XDATA_FIFO in addition to a TPM_DATA_FIFO.
 - A TPM SHALL accept any data transfer size up to and including the size indicated by DataTransferSizeSupport.

NOTE: This includes data transfers from 1-4 bytes, which may be written to either a TPM_DATA_FIFO or a TPM_XDATA_FIFO.

Table 30 — Interface Capability

Abbreviation:	TPM_INTF_CAPABILITY_x
----------------------	-----------------------

General Description:			Provides information about which interrupts this TPM supports. A TPM SHALL implement this register.	
Bit Descriptions:				
31	Read Only	Reserved	Default: Vendor Defined	The value of this field is not specified.
30:28	Read Only	InterfaceVersion	Default: Defined by hardware	000: Interface 1.21 or earlier 001: Reserved 010: Interface 1.3 011: Interface 1.3 for TPM 2.0 as defined in this specification. 100-111: Reserved
27:11	Read Only	Reserved	Reads always return 0	
10:9	Read Only	DataTransferSizeSupport	Default: 00 but defined by hardware	Indicates what transaction size a TPM supports for Data transfers: 11 = TPM supports 64-byte maximum transfer size (note, support for 64-byte transfer size also indicates support for legacy, 8- and 32-byte transfers) 10 = TPM supports 32-byte maximum transfer size (includes support for legacy and 8-byte transfers) 01 = TPM supports 8-byte maximum transfer size (includes support for legacy) 00 = TPM supports legacy transfer size only
8	Read only	BurstCountStatic	Default: Defined by hardware	Indicates whether a TPM_STS_x.burstCount field is dynamic or static 1 = TPM_STS_x.burstCount is static 0 = TPM_STS_x.burstCount is dynamic
7	Read only	CommandReadyIntSupport	Default: Defined by hardware	Corresponds to TPM_INT_ENABLE_x.commandReadyEnable 1 = supported 0 = not supported
6	Read Only	InterruptEdgeFalling	Default: Defined by hardware	Falling edge interrupt support 1 = supported 0 = not supported
5	Read Only	InterruptEdgeRising	Default: Defined by hardware	Rising edge interrupt support 1 = supported 0 = not supported
4	Read Only	InterruptLevelLow	Mandatory: SHALL be 1	Low level interrupt support. This interrupt trigger is mandatory. 1 = supported 0 = not allowed
3	Read Only	InterruptLevelHigh	Default: Defined by hardware	High level interrupt support 1 = supported 0 = not supported
2	Read Only	LocalityChangeIntSupport	Mandatory: SHALL be 1	Corresponds to TPM_INT_ENABLE_x.localityChangeIntEnable. This is a mandatory interrupt. 1 = supported 0 = not allowed

1	Read Only	stsValidIntSupport	Default: Defined by hardware	Corresponds to TPM_INT_ENABLE_x.stsValidIntEnable 1 = supported 0 = not supported
0	Read Only	dataAvailIntSupport	Mandatory: SHALL be 1	Corresponds to TPM_INT_ENABLE_x.dataAvailIntEnable. This is a mandatory interrupt. 1 = supported 0 = not allowed

6.5.2.8 Status Field State Transitions

Table 31 shows the changes in status fields based on the command or action done to a TPM. Notice this is not a state transition table covering the states defined in Section 6.5.2.5.1 TPM Status Register States, rather a table describing how the status fields change based on initial condition and action taken. The following rules apply to Table 31.

1. There MAY be intermediate status field states, where a command has finished, but TPM_STS_x.dataAvail is not yet set. Software is expected to poll until the appropriate status field is set.
2. The fields in the table represent values only when TPM_STS_x.stsValid = 1. Transitional states when TPM_STS_x.stsValid = 0 are neither captured nor represented in this table.
3. This table applies only to status field states where a locality has already been selected and no change in locality is performed.
4. The statements in the column labeled “Action Taken” are informative when shaded and are derived from normative statements contained within the definitions of a TPM_STS_x register in Section 6.5.2.5 Status Register, the description of the FIFO handling in Section 6.5.2.1 Normative 1.b Handling Command FIFOs, the description of the command transmission in Section 6.5.2.2.1 Command Send and the description of the response reception in Section 6.5.2.2.2 Data Availability. If there is an inconsistency between this column and the statements within normative definitions of a TPM_STS.x registers, the normative statements within definitions of a TPM_STS.x registers take precedence. The statements made in unshaded text and delimited by the phrases: “Start of normative comment” and “End of normative comment” are normative. Note that this is a reversal of the standard notation.
5. Normal transitions are highlighted in yellow and are indexed to the state transitions illustrated in Figure 3.
6. In all cases in Table 31 where Idle is the next state, a TPM is allowed to transition directly to the Ready state via transition 0.B. This simplifies the table by not having to show two ending states possibilities. When making a transparent transition to the Ready state, a TPM is not required to indicate it is or has been in the Idle state to the Software, therefore making this transition transparent to the Software.
7. The following abbreviations are used in Table 31:

Label	Bit Definition
C/R	TPM_STS_x.commandReady
D/A	TPM_STS_x.dataAvail
E	TPM_STS_x.Expect
TG	TPM_STS_x.tpmGo
R/R	TPM_STS_x.responseRetry
N/C	No Change to this field from previous state
—	No TPM access for the corresponding data element
X	Either 0 or 1. A TPM is allowed to maintain this field as either value for this state. Software must always be capable of managing a TPM, irrespective of the value.

Table 31 — State Transition Table

#	Present State of TPM_STS_x				Fields / Data Written to TPM_STS_x or TPM_DATA_FIFO.x				Next State & Result / Reason				Action Taken	
	TPM State	C/R	D/A	E	C/R	T	G/R	Write Data	Read Data	TPM State	C/R	D/A	E	Informative
0.A	Idle	0	0	0	—	—	—	—	—	Idle	0	0	0	<p>Start of normative comment t_r = Time since entering the Idle state If TPM is Idle and $t_r \geq \text{TIMEOUT}_B$ Then: TPM SHALL remain in the Idle state (i.e., continue executing in this in this row/state) until commanded to change by the Software (i.e., via state transition in row 3). Else: (i.e., $t_r < \text{TIMEOUT}_B$) TPM MAY transition to the Ready state (i.e., transition to row 0B)</p> <p>End of normative comment</p>
0.B	Idle	0	0	0	—	—	—	—	—	Ready	1	0	X	<p>This specification allows a TPM to be implemented such that the Software never sees the Idle state. This transition codifies and enables that behavior.</p> <p>Start of normative comment t_r = Time since entering the Idle state If TPM is Idle and $t_r \geq \text{TIMEOUT}_B$ Then: TPM SHALL NOT enter the Ready state (i.e., it SHALL NOT execute the state transition in this row and SHALL remain Idle in Row 0A). Else: (i.e., $t_r < \text{TIMEOUT}_B$) TPM MAY transition to the Ready state (i.e., execute the transition in this row).</p> <p>If a TPM performs this transition, it is not required to indicate that it is, or has been, in the Idle state; rather it is allowed to appear as if it went directly from the state prior to the Idle state to the Ready state transparently to the Software.</p> <p>End of normative comment</p>
1	Idle	0	0	0	0	0	1			Idle	0	0	0	There is no response to retry. No state changes.
2	Idle	0	0	0	0	1	0			Idle	0	0	0	There is no command to execute. No state changes.
3	Idle	0	0	0	1	0	0			Ready	1	0	X	This is the typical state change resulting from the Software's request to send a command.
4	Idle	0	0	0				Write data		Idle	0	0	0	TPM drops write, since it is not expecting data. If the Software was sending a command and still has data to write, then this is an error in transmission and the Software must re-transmit the command.
5	Idle	0	0	0					Read data	Idle	0	0	0	TPM returns FFh, since TPM_STS_x.dataAvail is not set.
6	Ready	1	0	X	0	0	1			Ready	1	0	X	No effect on TPM, since TPM_STS_x.commandReady indicates that the response has been read successfully.
7	Ready	1	0	X	0	1	0			Ready	1	0	X	Causes no change to TPM, since there is no command to process.
8	Ready	1	0	X	1	0	0			Ready	1	0	X	No effect on TPM, since it is ready to receive commands.

#	Present State of TPM_STS_x				Fields / Data Written to TPM_STS_x or TPM_DATA_FIFO.x				Next State & Result / Reason				Action Taken
	TPM State	C/R	D/A	E	C/R	TGR/R	Write Data	Read Data	TPM State	C/R	D/A	E	Informative
9	Ready	1	0	X			Write first byte		Reception	0	0	1	Clear TPM_STS_x.commandReady on first byte.
10	Ready	1	0	X				Read Data	Ready	1	0	X	No effect on TPM. TPM returns FFh.
11	Reception	0	0	1	0	0	1		Reception	0	0	1	There is no response to retry. No state changes.
12	Reception	0	0	1	0	1	0		Reception	0	0	1	TpmGo is not valid at this time. TPM ignores this state transition.
13	Reception	0	0	1	1	0	0		Idle	0	0	0	The command being sent to a TPM is aborted.
14	Reception	0	0	1			Write more data other than last byte		Reception	0	0	1	
15	Reception	0	0	1			Write last byte		Reception	0	0	0	Good transmission if the Software has just sent the last byte. If either Software has more than one more byte to send or if expect = 1 and Software has not more data to send, this is a transmission error, and the Software must resend the command.
16	Reception	0	0	1				Read data	Reception	0	0	1	TPM returns FFh.
17	Reception	0	0	0	0	0	1		Reception	0	0	0	No response to retry.
18	Reception	0	0	0	0	1	0		Execution	0	0	0	This is the normal transition from sending the command to the start of execution of the command.
19	Reception	0	0	0	1	0	0		Idle	0	0	0	Aborts the command sent.
20	Reception	0	0	0			Write		Reception	0	0	0	Write is not expected. Drop write. TPM ignores this state transition.
21	Reception	0	0	0				Read	Reception	0	0	0	Read 0xFF.
22	Execution	0	0	0	—	—	—	—	Completion	0	1	0	Upon command completion, TPM sets TPM_STS_x.dataAvail to a 1.
23	Execution	0	0	0	0	0	1		Execution	0	0	0	There is no response to retry. No state changes.
24	Execution	0	0	0	0	1	0		Execution	0	0	0	TPM is already executing a command. No state changes.
25	Execution	0	0	0	1	0	0		Idle	0	0	0	The executing command is aborted.
26	Execution	0	0	0			Write data		Execution	0	0	0	TPM drops write, since it is not expecting data. If the Software was sending a command and still has data to write, then this is an error in transmission and the Software must re-transmit the command.
27	Execution	0	0	0				Read data	Execution	0	0	0	TPM returns FFh, since TPM_STS_x.dataAvail is not set.
28	Completion	0	1	0	0	0	1		Completion	0	1	0	TPM resets ReadFIFO pointers and start sending the response from the first byte.
29	Completion	0	1	0	0	1	0		Completion	0	1	0	Causes no change to TPM, no new command to execute.
30	Completion	0	1	0	1	0	0		Idle	0	0	0	Aborts command.
31	Completion	0	1	0			Write data		Completion	0	1	0	No effect on TPM, it is not accepting a command.
32	Completion	0	1	0				Read data other than last byte	Completion	0	1	0	TPM returns data.

#	Present State of TPM_STS_x				Fields / Data Written to TPM_STS_x or TPM_DATA_FIFO.x				Next State & Result / Reason				Action Taken
	TPM State	C/R	D/A	E	C/R	TGR/R	Write Data	Read Data	TPM State	C/R	D/A	E	<i>Informative</i>
33	Completion	0	1	0				Read last byte	Completion	0	0	0	Good transmission, since TPM_STS_x.dataAvail = 0.
35	Completion	0	0	0	0	0	1		Completion	0	1	0	TPM resets ReadFIFO pointers and start sending the response from the first byte.
36	Completion	0	0	0	0	1	0		Completion	0	0	0	TpmGo is not valid at this time. TPM ignores this state transition.
37	Completion	0	0	0	1	0	0		Idle	0	0	0	This is the typical state change resulting from the Software's indication that it received the results of the command, and TPM may proceed to the Idle state and, depending on implementation, proceed directly to the Ready state.
38	Completion	0	0	0				Write	Completion	0	0	0	Write is not expected. Drop write. TPM ignores this state transition.
39	Completion	0	0	0				Read	Completion	0	0	0	Read 0xFF.
40	Any				More than 1 field set on any status write				undefined				If a write to this register contains more than one 1 field, the behavior of the status register is undefined in this specification and the behavior between TPM implementations may differ.

DRAFT

6.5.3 CRB Interface Requirements

6.5.3.1 CRB Addresses

Start of informative comment

A TPM compliant with this specification will implement the command and response buffers at the specified location for `TPM_CRB_DATA_BUFFER_x`. The CRB registers `TPM_CRB_CTRL_CMD_LADDR_x`, `TPM_CRB_CTRL_CMD_HADDR_x`, and `TPM_CRB_CTRL_RSP_ADDR_x` return the address of a `TPM_CRB_DATA_BUFFER_x` on read requests. For TPMs implemented to comply with other platform specifications that utilize the CRB interface, these registers may return addresses other than the PC Client defined address. For a PC Client compliant TPM, these fields will return the address defined in Table 32.

The CRB interface is not supported on the LPC physical interface.

The fields within the CRB Interface are naturally aligned so that 8-bit fields are located at 8-bit addressable locations and 64-bit fields are at 64-bit addressable locations. For this reason, the addresses for the Command Buffer are separated into two 32-bit fields, as it is not on a 64-bit addressable boundary. An 8-bit addressable location located on a 64-bit boundary may not be accessible using a 64-bit transaction.

Some instantiating hardware may have size and alignment restrictions when accessing any of the fields in this interface. Some hardware may require access to any of the data within a field or buffer be performed by reads or writes which are naturally aligned. For this reason, Software should access the fields and buffers defined in this interface using instructions which do not cause an access to cross an alignment boundary and should not use string move instructions.

For example:

Accessing the Response Address using a 64-bit access will work; accessing the Command Address using a 64-bit access may not work.

Note 1: The addresses defined in Table 32 — Address Allocation for CRB TPM Access assume a 4 KB page size for the host processor, as the offset for each locality begins at a 4KB boundary. A TPM implementation may support processors that utilize page sizes greater than 4KB by implementing the start offset for each locality at a boundary greater than 4KB. It is out of the scope of this specification how software determines the implemented page size.

Note 2: Discrete TPMs implemented in platforms with non-x86 chipsets might be located at base addresses other than `0xFED4_xxxx`. If implemented at a different base address, a `TPM_CRB_CMD_xADDR` registers have no meaning to a driver as the base address for the command buffer will be populated in a TPM object in the ACPI table. If device drivers use these registers, they should mask off the upper byte of `TPM_CRB_CMD_LADDR_x` as a TPM is unaware of the upper byte of the address and might report the value defined in this specification “0xFE”.

End of informative comment

Table 32 — Address Allocation for CRB TPM Access

Offset	Register Name	Description
Locality 0		
0003h-0000h	<code>TPM_LOC_STATE_0</code>	Used to determine current state of locality of a TPM. This register is aliased across all localities. Read-only.
0007h-0004h	Undefined	Reserved
000Bh-0008h	<code>TPM_LOC_CTRL_0</code>	Used to gain control of a TPM by this locality.
000Fh-000Ch	<code>TPM_LOC_STS_0</code>	Used to determine whether locality has been granted or seized. Read-only. This register SHALL NOT be aliased.
0013h-0010h	<code>TPM_DATA_CSUM_ENABLE_0</code>	Register to enable checksum computation on command and response
0017h-0014h	<code>TPM_DATA_CSUM_0</code>	Register to read the checksum computed
002Fh-0018h	Reserved	Reserved

Offset	Register Name	Description
0037h-0030h	TPM_CRB_INTF_ID_0	Used to identify the Interface types supported by a TPM as well as the Vendor ID, Device ID and Revision ID
003Fh-0038h	Reserved	Reserved
0043h-0040h	TPM_CRB_CTRL_REQ_0	Register used to initiate transactions for the CRB interface. This register may be aliased across localities.
0047h-0044h	TPM_CRB_CTRL_STS_0	Register used by a TPM to provide status of the CRB interface. This register may be aliased across localities
004Bh-0048h	TPM_CRB_CTRL_CANCEL_0	Register used by Software to cancel command processing. This register may be aliased across localities.
004Fh-004Ch	TPM_CRB_CTRL_START_0	Register used to indicate presence of command or response data in the CRB buffer. This register may be aliased across localities
0053h-0050h	TPM_CRB_INT_ENABLE_0	Register used to configure interrupts. This register may be aliased across localities
0057h-0054h	TPM_CRB_INT_STS_0	Register used to respond to interrupts. This register may be aliased across localities
005Bh-0058h	TPM_CRB_CTRL_CMD_SIZE_0	Size of the Command buffer. This register may be aliased across localities.
005Fh-005Ch	TPM_CRB_CTRL_CMD_LADDR_0	Lower 32bits of the Command buffer start address for Locality 0. This register may be aliased across localities.
0063h-0060h	TPM_CRB_CTRL_CMD_HADDR_0	Upper 32bits of the Command buffer start address for Locality 0. This register may be aliased across localities.
0067h-0064h	TPM_CRB_CTRL_RSP_SIZE_0	Size of the Response buffer. Note: If command and response buffers are implemented as a single buffer, this field SHALL be identical to the value in a TPM_CRB_CTRL_CMD_SIZE_x buffer. This register may be aliased across localities.
006Fh-0068h	TPM_CRB_CTRL_RSP_ADDR_0	Address of the start of the Response buffer. Note: If command and response buffers are implemented as a single buffer, this field SHALL contain the same address contained in TPM_CRB_CTRL_CMD_HADDR_x and TPM_CRB_CMD_LADDR_x. This register may be aliased across localities.
007Fh-0070h	Reserved	Reserved
0EFFh-0080h	TPM_CRB_DATA_BUFFER_0	Command/Response Data may be defined as large as 3968. This is implementation specific. However, the full address space has been reserved. This buffer may be aliased across localities. This field accepts data transfers from 1B up to the size indicated by TPM_CRB_INTF_ID_x.CapDataXferSizeSupport (see section 6.4.2.2 CRB Interface Identifier Register).
0FFFh-0F00h	Reserved	Reserved for maximum size of Command/Response Buffer
Locality 1		
1003h-1000h	TPM_LOC_STATE_1	Same as TPM_LOC_STATE_0
1007h-1004h	Reserved	Reserved
100Bh-1008h	TPM_LOC_CTRL_1	Used to gain control of a TPM by this locality.
100Fh-100Ch	TPM_LOC_STS_1	Used to determine whether locality has been granted or seized. Read-only. This register SHALL NOT be aliased.
0013h-0010h	TPM_DATA_CSUM_ENABLE_1	Same as TPM_DATA_CSUM_ENABLE_0
0017h-0014h	TPM_DATA_CSUM_1	Same as TPM_DATA_CSUM_0
102Fh-1018h	Reserved	Reserved

Offset	Register Name	Description
1037h-1030h	TPM_CRB_INTF_ID_1	Same as TPM_CRB_INTF_ID_0
103Fh-1038h	Reserved	Reserved
1043h-1040h	TPM_CRB_CTRL_REQ_1	Same as TPM_CRB_CTRL_REQ_0
1047h-1044h	TPM_CRB_CTRL_STS_1	Same as TPM_CRB_CTRL_STS_0
104Bh-1048h	TPM_CRB_CTRL_CANCEL_1	Same as TPM_CRB_CTRL_CANCEL_0
104Fh-104Ch	TPM_CRB_CTRL_START_1	Same as TPM_CRB_CTRL_START_0
1053h-1050h	TPM_CRB_INT_ENABLE_1	Same as TPM_CRB_INT_ENABLE_0
1057h-1054h	TPM_CRB_INT_STS_1	Same as TPM_CRB_INT_STS_0
105Bh-1058h	TPM_CRB_CTRL_CMD_SIZE_1	Same as TPM_CRB_CTRL_CMD_SIZE_0
105Fh-105Ch	TPM_CRB_CTRL_CMD_LADDR_1	Same as TPM_CRB_CTRL_CMD_LADDR_0
1063h-1060h	TPM_CRB_CTRL_CMD_HADDR_1	Same as TPM_CRB_CTRL_CMD_HADDR_0
1067h-1064h	TPM_CRB_CTRL_RSP_SIZE_1	Same as TPM_CRB_CTRL_RSP_SIZE_0
106Fh-1068h	TPM_CRB_CTRL_RSP_ADDR_1	Same as TPM_CRB_CTRL_RSP_ADDR_0
107Fh-1070h	Reserved	Reserved
1EFFh-1080h	TPM_CRB_DATA_BUFFER_1	Same as TPM_CRB_DATA_BUFFER_0
1FFFh-1F00h	Reserved	Reserved for maximum size of Command/Response Buffer
Locality 2		
2003h-2000h	TPM_LOC_STATE_2	Same as TPM_LOC_STATE_0
2007h-2004h	Reserved	Reserved
200Bh-2008h	TPM_LOC_CTRL_2	Used to gain control of a TPM by this locality.
200Fh-200Ch	TPM_LOC_STS_2	Used to determine whether locality has been granted or seized. Read-only. This register SHALL NOT be aliased.
0013h-0010h	TPM_DATA_CSUM_ENABLE_2	Same as TPM_DATA_CSUM_ENABLE_0
0017h-0014h	TPM_DATA_CSUM_2	Same as TPM_DATA_CSUM_0
202Fh-2018h	Reserved	Reserved
2037h-2030h	TPM_CRB_INTF_ID_2	Same as TPM_CRB_INTF_ID_0
203Fh-2038h	Reserved	Reserved
2043h-2040h	TPM_CRB_CTRL_REQ_2	Same as TPM_CRB_CTRL_REQ_0
2047h-2044h	TPM_CRB_CTRL_STS_2	Same as TPM_CRB_CTRL_STS_0
204Bh-2048h	TPM_CRB_CTRL_CANCEL_2	Same as TPM_CRB_CTRL_CANCEL_0
204Fh-204Ch	TPM_CRB_CTRL_START_2	Same as TPM_CRB_CTRL_START_0
2053h-2050h	TPM_CRB_INT_ENABLE_2	Same as TPM_CRB_INT_ENABLE_0
2057h-2054h	TPM_CRB_INT_STS_2	Same as TPM_CRB_INT_STS_0
205Bh-2058h	TPM_CRB_CTRL_CMD_SIZE_2	Same as TPM_CRB_CTRL_CMD_SIZE_0
205Fh-205Ch	TPM_CRB_CTRL_CMD_LADDR_2	Same as TPM_CRB_CTRL_CMD_LADDR_0
2063h-2060h	TPM_CRB_CTRL_CMD_HADDR_2	Same as TPM_CRB_CTRL_CMD_HADDR_0
2067h-2064h	TPM_CRB_CTRL_RSP_SIZE_2	Same as TPM_CRB_CTRL_RSP_SIZE_0
206Fh-2068h	TPM_CRB_CTRL_RSP_ADDR_2	Same as TPM_CRB_CTRL_RSP_ADDR_0
207Fh-2070h	Reserved	Reserved
2EFFh-2080h	TPM_CRB_DATA_BUFFER_2	Same as TPM_CRB_DATA_BUFFER_0
2FFFh-2F00h	Reserved	Reserved for maximum size of Command/Response Buffer

Locality 3		
3003h-3000h	TPM_LOC_STATE_3	Same as TPM_LOC_STATE_0

3007h-3004h	Reserved	Reserved
300Bh-3008h	TPM_LOC_CTRL_3	Used to gain control of a TPM by this locality.
300Fh-300Ch	TPM_LOC_STS_3	Used to determine whether locality has been granted or seized. Read-only. This register SHALL NOT be aliased.
0013h-0010h	TPM_DATA_CSUM_ENABLE_3	Same as TPM_DATA_CSUM_ENABLE_0
0017h-0014h	TPM_DATA_CSUM_3	Same as TPM_DATA_CSUM_0
302Fh-3018h	Reserved	Reserved
3037h-3030h	TPM_CRB_INTF_ID_3	Same as TPM_CRB_INTF_ID_0
303Fh-3038h	Reserved	Reserved
3043h-3040h	TPM_CRB_CTRL_REQ_3	Same as TPM_CRB_CTRL_REQ_0
3047h-3044h	TPM_CRB_CTRL_STS_3	Same as TPM_CRB_CTRL_STS_0
304Bh-3048h	TPM_CRB_CTRL_CANCEL_3	Same as TPM_CRB_CTRL_CANCEL_0
304Fh-304Ch	TPM_CRB_CTRL_START_3	Same as TPM_CRB_CTRL_START_0
3053h-3050h	TPM_CRB_INT_ENABLE_3	Same as TPM_CRB_INT_ENABLE_0
3057h-3054h	TPM_CRB_INT_STS_3	Same as TPM_CRB_INT_STS_0
305Bh-3058h	TPM_CRB_CTRL_CMD_SIZE_3	Same as TPM_CRB_CTRL_CMD_SIZE_0
305Fh-305Ch	TPM_CRB_CTRL_CMD_LADDR_3	Same as TPM_CRB_CTRL_CMD_LADDR_0
3063h-3060h	TPM_CRB_CTRL_CMD_HADDR_3	Same as TPM_CRB_CTRL_CMD_HADDR_0
3067h-3064h	TPM_CRB_CTRL_RSP_SIZE_3	Same as TPM_CRB_CTRL_RSP_SIZE_0
306Fh-3068h	TPM_CRB_CTRL_RSP_ADDR_3	Same as TPM_CRB_CTRL_RSP_ADDR_0
307Fh-3070h	Reserved	Reserved
3EFFh-3080h	TPM_CRB_DATA_BUFFER_3	Same as TPM_CRB_DATA_BUFFER_0
3FFFh-3F00h	Reserved	Reserved for maximum size of Command/Response Buffer
Locality 4		
4003h-4000h	TPM_LOC_STATE_4	Same as TPM_LOC_STATE_0
4007h-4004h	Undefined	Reserved
400Bh-4008h	TPM_LOC_CTRL_4	Used to gain control of a TPM by this locality.
400Fh-400Ch	TPM_LOCALITY_STS_4	Used to determine whether locality has been granted or seized. Read-only. This register SHALL NOT be aliased.
0013h-0010h	TPM_DATA_CSUM_ENABLE_4	Same as TPM_DATA_CSUM_ENABLE_0
0017h-0014h	TPM_DATA_CSUM_4	Same as TPM_DATA_CSUM_0
402Fh-4018h	Undefined	Reserved
4037h-4030h	TPM_CRB_INTF_ID_4	Same as TPM_CRB_INTF_ID_0
403Fh-4038h	Undefined	Reserved
4043h-4040h	TPM_CRB_CTRL_REQ_4	Same as TPM_CRB_CTRL_REQ_0
4047h-4044h	TPM_CRB_CTRL_STS_4	Same as TPM_CRB_CTRL_STS_0
404Bh-4048h	TPM_CRB_CTRL_CANCEL_4	Same as TPM_CRB_CTRL_CANCEL_0
404Fh-404Ch	TPM_CRB_CTRL_START_4	Same as TPM_CRB_CTRL_START_0
4053h-4050h	TPM_CRB_INT_ENABLE_4	Same as TPM_CRB_INT_ENABLE_0
4057h-4054h	TPM_CRB_INT_STS_4	Same as TPM_CRB_INT_STS_0
405Bh-4058h	TPM_CRB_CTRL_CMD_SIZE_4	Same as TPM_CRB_CTRL_CMD_SIZE_0
405Fh-405Ch	TPM_CRB_CTRL_CMD_LADDR_4	Same as TPM_CRB_CTRL_CMD_LADDR_0
4063h-4060h	TPM_CRB_CTRL_CMD_HADDR_4	Same as TPM_CRB_CTRL_CMD_HADDR_0
4067h-4064h	TPM_CRB_CTRL_RSP_SIZE_4	Same as TPM_CRB_CTRL_RSP_SIZE_0
406Fh-4068h	TPM_CRB_CTRL_RSP_ADDR_4	Same as TPM_CRB_CTRL_RSP_ADDR_0
407Fh-4070h	Undefined	Reserved
4EFFh-4080h	TPM_CRB_DATA_BUFFER_4	Same as TPM_CRB_DATA_BUFFER_0

4FFFh-4F00h	Reserved	Reserved for maximum size of Command/Response Buffer
Non-locality Specific Registers		
5FFFh-5000h	Reserved	Reserved
All addresses not defined in the table above	Reserved, reads return FFh; writes are dropped.	

6.5.3.2 Locality Support

Start of informative comment

The concept of locality, as described in Section 5.3 Locality-Controlled Functions, is interface agnostic, but the registers used to interact with a TPM at various localities are necessarily different. This section describes the registers used to request and use a TPM at various localities.

End of informative comment

6.5.3.2.1 Locality State Register

Start of informative comment

This register is a read-only register used by host Software to determine the current state of a TPM with respect to locality. This register leverages some of the functionality from the FIFO ACCESS register. This register is aliased across all localities.

This register gates access to a TPM_CRB_CTRL_x and TPM_CRB_DATA_BUFFER_x registers. The default state of this register at power on of a TPM has no locality active. This results in any writes to the Control Area or Data Buffer being dropped.

End of informative comment

1. A TPM SHALL implement a TPM_LOC_STATE register as defined in Table 33.
2. The initial state of this register at power on SHALL clear a TPM_LOC_STATE_x.locAssigned field to 0 and TPM_LOC_STATE_x.activeLocality field to 000.
3. TPM_LOC_STATE_x.tpmEstablished SHALL be valid when tpmRegValidSts is set to 1.
 - a. TPM_LOC_STATE_x.tpmEstablished SHALL be set to 1 when TPM_LOC_CTRL_x.resetEstablishmentBit is set to 1
 - b. TPM_LOC_STATE_x.tpmEstablished SHALL be cleared to 0 upon receipt of a HASH_END.
4. A TPM SHALL NOT set TPM_LOC_STATE_x.tpmRegValidSts to 1 unless all other fields are valid.

Table 33 — TPM_LOC_STATE Definition

Abbreviation:			TPM_LOC_STATE_x	
General Description:			Used to determine status of the locality controls of a TPM.	
Bit Descriptions:				
31:8	Read Only	Reserved	0	Reserved: Reads return 0.
7	Read Only	tpmRegValidSts	0	This bit indicates that all other bits of this register contain valid values, when it is a 1.
6:5	Read Only	Reserved	0	Reserved: Reads return 0.
4:2	Read Only	activeLocality	0	000 – Locality 0 001 – Locality 1 010 – Locality 2 011 – Locality 3 100 – Locality 4 101:111 – Reserved: Reads return 0: This bit field informs host Software which locality currently has access to a TPM.
1	Read Only	locAssigned	0	A 0 indicates to the host that no locality is assigned, a 1 indicates a locality has been assigned.
0	Read Only	tpmEstablished	1	A TPM clears this bit to 0 upon receipt of HASH_END A TPM sets this bit to a 1 when a TPM_LOC_CTRL_x.resetEstablishment field is set to 1.

6.5.3.2.2 Locality Control Register

Start of informative comment

A TPM_LOC_CTRL register is used by each locality to request use of or seize control of a TPM. This register is unique for each locality. This register is defined to be read/write capable.

The functionality described in this register is specific to Localities 0-3. There are unique requirements for Locality 4.

For the caller to send a command to a TPM, it must first request use of a TPM through the locality control method. If locality is not granted, the CRB data buffer drops any command.

Note: A write to a TPM_LOC_CTRL_x register with more than one field set to 1 for the same Locality may be ignored by a TPM. Alternatively, a TPM may ignore Relinquish if both TPM_LOC_CTRL_x.Relinquish and TPM_LOC_CTRL_x.requestAccess are set to 1 for the same Locality. If a TPM sees a request for multiple localities while a TPM is being used by another locality, a TPM will honor the request from the highest locality once the controlling locality relinquishes control. See section 5.3 Locality-Controlled Functions for a complete description of the Locality arbitration process.

TPMs implemented in SoCs or in memory do not have full control over this register in memory. When a caller notifies a TPM that data has been written to this register, a TPM will read the memory and needs to zero the register once it has read it.

End of informative comment

6.5.3.2.2.1 Locality Control Register for Localities 0-3

1. A TPM SHALL implement a TPM_LOC_CTRL register as defined in Table 34.

Table 34 — TPM_LOC_CTRL_x Register Definition

Abbreviation:		TPM_LOC_CTRL_x		
General Description:		Used to gain control of a TPM		
Bit Descriptions:				
31:4	Write Only	Reserved	0	Reserved. Reads return 0.
3	Write Only	resetEstablishmentBit	0	Reads always return 0 Writes (0): Ignored Writes (1): Reset TPM_LOC_STATE_x.tpmEstablished bit if the write occurs from Locality 3 or 4. Valid indicator: NA
2	Write Only	Seize	0	Reads always return 0 Writes (0): Ignored Writes (1): A TPM gives control of a TPM to the locality setting this bit if it is the higher priority locality.
1	Write Only	Relinquish	0	Reads always return 0 Writes (0): Ignored Writes (1): The active locality is done with a TPM.
0	Write Only	requestAccess	0	Reads always return 0 Writes (0): Ignored. Writes (1): Interrupt a TPM and execute a locality arbitration algorithm. Note: This field corresponds to a TPM_ACCESS_x.requestUse field in the FIFO implementation.

6.5.3.2.2.2 Locality Control Register for Locality 4

Table 35 — TPM_LOC_CTRL_4 Register Definition

Abbreviation:			TPM_LOC_CTRL_4	
General Description:			Used to perform actions of D-RTM Sequence	
Bit Descriptions:				
31:4	Write Only	Reserved	0	Reads return 0h
3	Write Only	resetEstablishment	0	Reads always return 0 Writes (0): Ignored Writes (1): Reset TPM_LOC_STATE_x.tpmEstablished bit if the write occurs from Locality 3 or 4. Valid indicator: NA
2	Write Only	TPM_HASH_END	0	Reads return 0h Writes (0): Ignored Writes (1): Initiates the HASH_END TPM actions (see Section 5.3 Locality-Controlled Functions)
1	Write Only	TPM_HASH_DATA	0	Reads return 0h Writes (0): Ignored Writes (1): Initiates the HASH_DATA TPM actions (see Section 5.3 Locality-Controlled Functions)
0	Write Only	TPM_HASH_START	0	Reads return 0h Writes (0): Ignored Writes (1): Initiates the HASH_START TPM actions (see Section 5.3 Locality-Controlled Functions)

Bit Field: resetEstablishmentBit

Start of informative comment

This field is used to replace functionality provided by the TSC_ResetEstablishmentBit command in TPM 1.2. This command is not present in TPM 2.0, but this interface mechanism provides equivalent functionality.

This field is used to reset the state of a TPM_LOC_STATE_x.tpmEstablished bit, once that bit has been set to 0 by a D-RTM sequence. A write to this field will be processed by a TPM as if a command has been received. Once this field is written, a TPM will clear TPM_CRB_CTRL_STS_x.cmdReady until a TPM_LOC_STATE_x.tpmEstablished bit has been cleared.

This field can only be written from Localities 3 and 4.

End of informative comment

1. Reads to TPM_LOC_CTRL_x.resetEstablishmentBit SHALL always return 0.
2. For Localities 0-2, writes are ignored.
3. For Localities 3 and 4
 - a. Writes of 0 are ignored.
 - b. On a write of 1, when in the Ready or Idle state a TPM SHALL set TPM_LOC_STATE_x.tpmEstablished to 1 and clear TPM_LOC_CTRL_x.resetEstablishmentBit to 0 within TIMEOUT_A, see Section 6.5.1.4 Interface Timeouts.

Field: TPM_HASH_START/_DATA/_END

Start of informative comment

These fields are used to instantiate an S-HCRTM or D-RTM sequence as defined in Section 5.3 Locality-Controlled Functions. There is no data placed in these fields. The data is placed in the command/response buffer. The normative behavior for these fields is documented in section 5.3.1 D-RTM Execution Sequence.

Note: For the CRB interface, an optimization allows both a TPM_HASH_DATA and TPM_HASH_END fields of a TPM_LOC_CTRL_4 register to be SET in the same write cycle.

End of informative comment

1. A write to TPM_LOC_CTRL_4.TPM_HASH_START SHALL invoke the _TPM_Hash_Start interface command as defined in a TPM 2 Library Specification and perform the actions of HASH_START in Section 5.3.1 D-RTM Execution Sequence.
2. A write to TPM_LOC_CTRL_4.TPM_HASH_DATA SHALL invoke the _TPM_Hash_Data interface command as defined in a TPM 2 Library Specification and perform the actions of HASH_DATA in Section 5.3.1 D-RTM Execution Sequence.
3. A write to TPM_LOC_CTRL_4.TPM_HASH_END SHALL invoke the _TPM_Hash_End interface command as defined in a TPM 2 Library Specification and perform the actions of HASH_END in Section 5.3.1 D-RTM Execution Sequence.

6.5.3.2.2.3 LOCALITY_STATUS Register

Table 36 —TPM_LOC_STS

Abbreviation:		TPM_LOC_STS_x		
General Description:				
Bit Descriptions:				
31:2	Read Only	Reserved	0	Reads return 0
1	Read Only	beenSeized	0	0: A higher locality has not initiated a Seize arbitration process. 1: A higher locality has Seized a TPM from this locality.
0	Read Only	Granted	0	0: Locality has not been granted to a TPM. 1: Locality has been granted access to a TPM

This register is unique for each locality.

6.5.3.3 Control Area Request Register

Start of informative comment

The Control Area Request register is used to manage TPM states as defined in Section 6.5.3.7 Interface Controls.

End of informative comment

Table 37 — TPM CRB Control Area Request

Abbreviation:			TPM_CRB_CTRL_REQ_x
General Description:			Control Area Request
Bit Descriptions:			
31:2	Read/Write	Reserved	Reserved. Read's return 0. Writes are ignored.
1	Read / Write	goldle	0 Used by Software to indicate transition of a TPM to and from the Idle state. 1: Set by Software to indicate response has been read from the response buffer and TPM can transition to Idle 0: Cleared to 0 by TPM to acknowledge completion of the state change request to the Idle state with the resulting state reflected in a TPM_CRB_STS_x.tpmlIdle field. TPM SHALL complete this transition within TIMEOUT_C. Writes of 0 are ignored.
0	Read / Write	cmdReady	0 Used by Software to request a TPM to transition to the Ready State. 1: Set to 1 by Software to indicate a TPM should be ready to receive a command. 0: Cleared to 0 by TPM to acknowledge completion of the state change request to the Idle state with the resulting state reflected in a TPM_CRB_STS_x.tpmlIdle field. TPM SHALL complete this transition within TIMEOUT_C Writes of 0 are ignored.

*Field cmdReady***Start of informative comment**

The cmdReady field is analogous to the FIFO TPM_ACCESS_x.cmdReady field. It is written by Software to transition a TPM to the Ready State from the Idle State.

When in the Idle State, a TPM may perform background tasks, such as random number or key generation, or it may turn off some of its internal functions to conserve power. Because of the non-deterministic nature of what a TPM may be doing in the background, it is necessary to define a time within which a TPM must respond to a request to transition to the Ready state. This time, TIMEOUT_C, see Section 6.5.1.4 Interface Timeouts, is the maximum amount of time a driver will need to wait before determining a TPM or its interface have encountered an error. TPM manufacturers should implement their TPMs to transition to the Ready State as quickly as possible.

In this version of this specification, a new, optional behavior was introduced. There may be cases where Software has a queue of commands to issue to a TPM. Some TPM implementations are limited in their ability to quickly transition in and out of the Idle state. These TPMs may now support the ability to bypass the Idle and Ready states and transition directly to the Ready state or Command Reception state from Command Completion. The ability to support this feature is reported in the field TPM_CRB_INTF_ID.CapCRBIdleBypass. If this field is set to 1, calling Software may exercise this new behavior by writing this field from the Command Completion state and a TPM will transition directly to the Ready state. As there is no interface mechanism to distinguish between the Command Completion and Ready states, calling Software may need to write this field twice to ensure a TPM is in the Ready state. TPM implementations that do not support this behavior are anticipated to perform the transition from Command Completion to Idle and from Idle to Ready in a shorter time frame than the sum of the timeouts for both state transitions. As TPM implementations use the Idle state to perform background processing that improves overall performance and

reduce power consumption, Software should not exercise this behavior unless there are multiple TPM commands in their queue.

End of informative comment

1. Reads to this field SHALL comply with the requirements in Section 7.6.
2. If a TPM is in Idle:
 - a. On a write of 1 to this field, a TPM SHALL transition from Idle to Ready State.
 - b. A TPM SHALL acknowledge the write by clearing the field to 0 within TIMEOUT_C, see Section 6.5.1.4 Interface Timeouts.
 - c. Writes of 0 are ignored.
3. If a TPM is in Command Completion:
 - a. On a write of 1:
 - i. If TPM_CRB_INTF_IDx.CapCRBIdleBypass is 1, a TPM SHALL transition to Ready within TIMEOUT_C, see Section 6.5.1.4 Interface Timeouts.
 - ii. If TPM_CRB_INTF_IDx.CapCRBIdleBypass is 0, a TPM SHALL ignore the write.
 - b. Writes of 0 are ignored.

Field goldle

Start of informative comment

When in the Command Completion state, Software will use this field to communicate to a TPM that it has received all the response data and a TPM may invalidate its buffer and transition to Idle.

There are occasions where an active transaction needs to be aborted. The driver and TPM may get out of synchronization. There may be a power transition that occurs prior to a TPM entering Command Execution. In these cases, it is desirable to have a mechanism to cause a TPM to transition back to the Idle state. This operation is not the same as a FIFO command abort, see Section 6.5.2.3.1 Command Aborts, but provides a limited approximation of that functionality.

Note: When in the Command Execution phase, a transaction cannot be aborted. The only option to interrupt the processing of the current command is to perform a Command Cancel.

End of informative comment

1. Reads to this field SHALL comply with the requirements in Section 7.6.
2. On a write to this field of 1, a TPM SHALL acknowledge the write by clearing the field to 0 within TIMEOUT_C.
3. On a write of 1, a TPM SHALL invalidate the Command/Response buffer and transition to Idle and set TPM_CRB_CTRL_STS_x.tpmlidle to 1.
4. Once a response is placed in the Command/Response buffer, a TPM SHALL maintain the response until Software sets this field to 1.
5. When a TPM is in the Ready, Command Reception, or Command Completion states, if TPM_CRB_CTRL_REQ_x.goldle is set to 1:
 - a. A TPM SHALL invalidate the command/response buffer,
 - b. A TPM SHALL set a TPM_CRB_CTRL_STS_x to 0002h, and then.
 - c. A TPM SHALL clear a TPM_CRB_CTRL_REQ_x field to 0000h.
6. When a TPM is in the Command Execution state, a TPM SHALL ignore writes to a TPM_CRB_CTRL_REQ_x field.
7. Writes of 0 are ignored.

6.5.3.4 Control Area Status Register

Start of informative comment

This register is used to indicate the current state and status of a TPM.

End of informative comment**Table 38 — TPM CRB Control Area Status**

Abbreviation:		TPM_CRB_CTRL_STS_x		
General Description:		Control Area Status		
Bit Descriptions:				
31:2	Read Only	Reserved		Reserved. Read's SHALL return 0's.
2	Read Only	cSUMAvailable	0	Used by a TPM to indicate the command checksum is available to be read by the host, if the checksum is supported. 1: Set by a TPM when the command CSUM is available 0: Indicates no CSUM is available See Section 6.5.1.8.2 TPM_DATA_CSUM
1	Read Only	tpmIdle	0	Used by the TPM to indicate it is in the Idle State 1: Set by the TPM when in the Idle State 0: Cleared by the TPM on receipt of TPM_CRB_CTRL_REQ_x.cmdReady when TPM transitions to the Ready State. SHALL be cleared by TIMEOUT_C.
0	Read Only	tpmSts	0	Used by the TPM to indicate current status. 1: Set by the TPM to indicate a FATAL Error 0: Indicates the TPM is operational

*Field: tpmIdle***Start of informative comment**

A TPMIdle field is used by a TPM to indicate its current state. A TPM sets this field to indicate that it is in the Idle state and clears it to indicate it is in the Ready State.

The default state of this field indicates a TPM is in the Idle State.

End of informative comment

1. Reads to this field SHALL comply with the requirements in Section 7.6.
2. A TPM SHALL set this field to 1 if going to Idle state.
3. A TPM SHALL NOT set this field to 1 if in any state other than Idle.
4. On a write to TPM_CRB_CTRL_REQ_x.cmdReady of 1, a TPM SHALL clear this field and go to the Ready state within TIMEOUT_C

*Field: tpmSts***Start of informative comment**

A TPMSts field is used by a TPM to indicate its current status. This field should only be used for fatal errors for which a TPM response code cannot be provided. An example is the response buffer is not in physical memory. Software may stop using a TPM when this field is set. The initial state of this field should reflect the operational state of a TPM.

Start of informative comment

1. Reads to this field SHALL be valid as defined in Section 7.6.

2. A TPM MAY set this field to 1 on a fatal error.
3. A TPM SHALL NOT set this field to 1 for any other reason other than a fatal error.

6.5.3.5 Control Area Cancel Register

Start of informative comment

Cancel may be used by Software to request a TPM to terminate processing the current command. Software might request this because the system is going to a lower power state. A TPM will either complete the currently processing command and return the result or will cancel the command and return the return code TPM_RC_Canceled. In general, for long running commands, a TPM may have checkpoints in its code to check the state of the Cancel field. If, at one of these checkpoints, a TPM sees a Command Cancel request, a TPM has the option of canceling the command or completing the command. TPMs are not required to perform this check.

Cancel is an asynchronous input. Driver writers should take care in use of this function in their drivers, as a TPM could enter a state where it would process no commands. This case may occur if Software sends a Cancel but fails to clear it. A TPM may be implemented in such that it will continue to cancel or complete the command and transition to Command Completion with a result. Alternatively, TPMs may be implemented such that they only cancel a single command. Driver writers should take care to send Command Cancel requests only when a TPM is in Command Execution state and to only clear the field when a TPM has exited Command Execution to avoid race conditions.

When a TPM is in Command Execution, and it receives a Seize request, a TPM will treat Seize as a Command Cancel followed by a transition to Idle. When a TPM is not in Command Execution, Software may write this field at any time.

End of informative comment

Table 39 — TPM CRB Control Cancel

Abbreviation:		TPM_CRB_CTRL_CANCEL_x		
General Description:		Control Area Cancel		
Bit Descriptions:				
31:0	Read / Write	Cancel	0000 0000h	Used by Software to cancel command processing. Reads return correct value. Writes (0000 0001h): Cancel a command. Writes (0000 0000h): Clears field when command has been cancelled

1. When a TPM is in the Command Execution state and this field is set to 0001h:
 - a. A TPM MAY terminate command processing,
 - b. A TPM SHALL return a response and transition to the Command Completion state:
 - i. If the command is successfully completed, a TPM SHALL return the command response.
 - ii. If the command is terminated, a TPM SHALL return TPM_RC_CANCELED.
 - iii. A TPM SHALL complete the command or cancel it within TIMEOUT_B.
 - iv. A TPM SHALL clear the Start field to 0000 0000h.
2. When a TPM is in any state other than Command Execution, a TPM SHOULD ignore writes to this field.

6.5.3.6 Control Area Start Register

Start of informative comment

Start is used by Software to signal a TPM to begin processing the contents of the Command/Response Buffer and transition a TPM into Command Execution. When a TPM has completed or cancelled a command, a TPM clears this field and transitions to Command Completion.

Software should poll on this field to detect a TPM transitioning from Command Execution to Command Completion.

Note: Some platforms may not be able to trigger TPM commands based on the start field in the control area and may require the implementation of an optional ACPI Start method to allow Software to request the system to execute a

TPM command. The use of the ACPI Start method is determined by the Start Method field of a TPM2 ACPI table defined in the TCG ACPI Specification [11].

End of informative comment

Table 40 — TPM CRB Control Start

Abbreviation:		TPM_CRB_CTRL_START_x	
General Description:		Control Area Start	
Bit Descriptions:			
31:0	Read / Write	Start	0000 0000h
			When set by Software, indicates a command is ready for processing. A write of 0000 0001h: TPM transitions to Command Execution A write of 0000 0000h: TPM clears this field and transitions to Command Completion

1. On a write of 0001h to the Start field:
 - a. If a TPM is in the Command Reception state, a TPM SHALL transition to Command Execution and begin processing the command in the buffer.
 - b. If a TPM is in any other State, a TPM SHALL ignore the write.
2. A TPM SHALL ignore writes of 0 to this field.
3. When a TPM completes command processing, a TPM SHALL clear this field to 0000h and transition to Command Completion.

6.5.3.7 Interface Controls

A TPM is in one of the following defined states:

1. *Command Reception* occurs following a *Ready* state between the write of the first byte of a command to the Command Buffer and the receipt of a write of 1 to TPM_CRB_CTRL_START_x.
2. *Command Execution* occurs after receipt of a 1 to TPM_CRB_CTRL_START_x and a TPM clearing TPM_CRB_CTRL_START_x to 0.
3. *Command Completion* occurs after completion of a command (indicated by a TPM clearing TPM_CRB_CTRL_Start_x to 0) and before a write of a 1 by Software to TPM_CRB_CTRL_REQ_x.goldle.
4. *Idle* is any time TPM_CRB_CTRL_STS_x.tpmlidle is 1. This will occur when a TPM is in *Command Completion* on receipt of a write of 1 by Software to TPM_CRB_CTRL_REQ_x.goldle and is signaled by a TPM setting TPM_CRB_CTRL_REQ_x.tpmlidle to 1. This will also occur following locality change. *Idle* is the initial state of TPM upon completion of _TPM_INIT. A TPM may skip this state on receipt of TPM_CRB_CTRL_REQ_x.cmdReady while in *Command Completion*.
5. *Ready* is any time a TPM is ready to receive a command. A TPM transitions from the *Idle* state to the *Ready* state by clearing TPM_CRB_CTRL_STS_x to 0 after receiving a write of 1 by Software to TPM_CRB_CTRL_STS_x.cmdReady, as indicated by the Status field being cleared to 0. If TPM_CRB_INTF_ID.CapCRBIdleBypass is set to 1, a TPM transitions from *Command Completion* state to the *Ready* state after clearing TPM_CRB_CTRL_REQ_x.cmdReady to 0 after receiving a write of 1 to TPM_CRB_CTRL_REQ_x.cmdReady.

Start of informative comment

The following informative diagram is derived from the above normative statements. It is informative and only for illustrating diagrammatically the above TPM states and their transitions. The numbers in parentheses reference the states represented by row numbers in Table 41 — CRB Interface State Transitions.

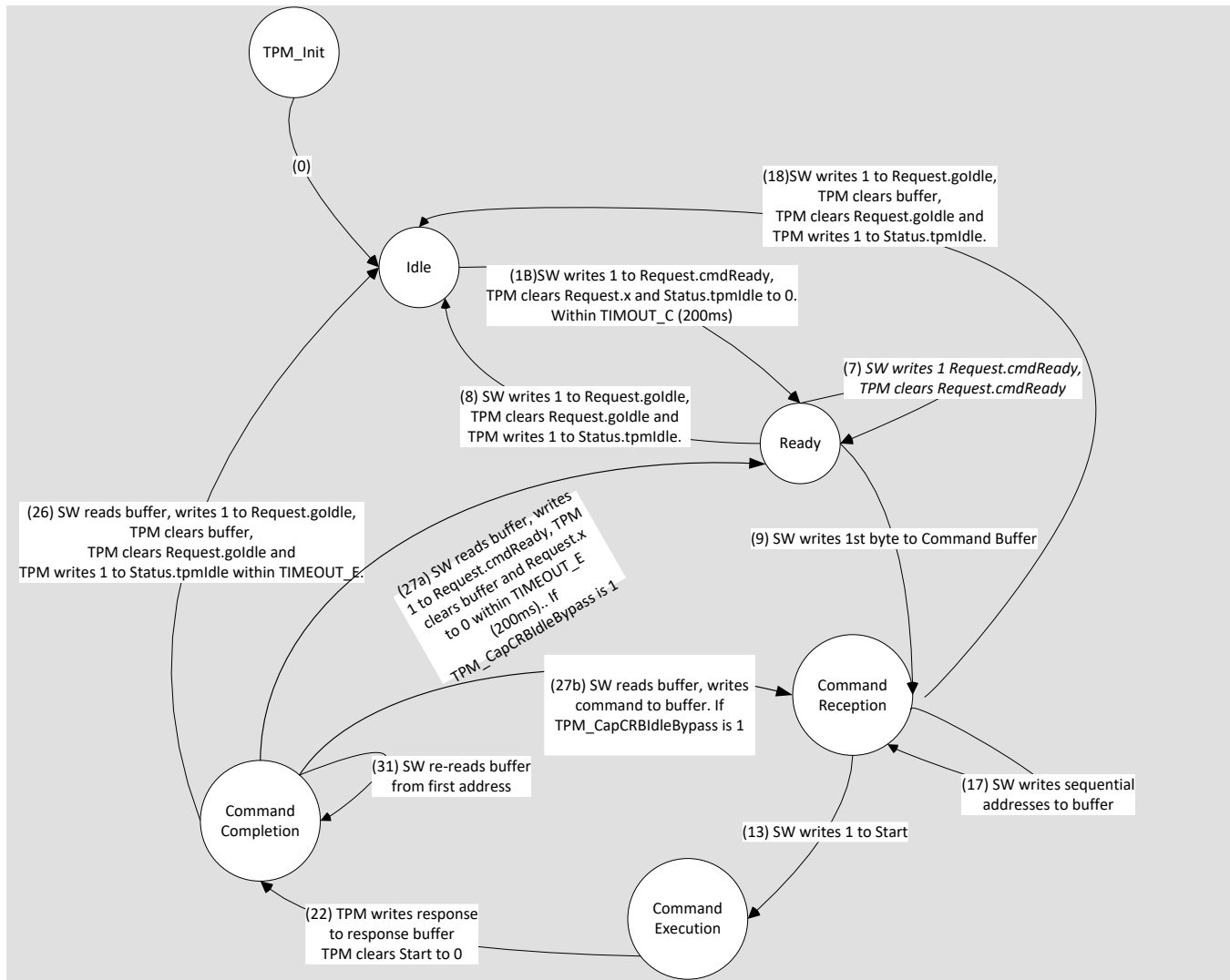


Figure 4 — TPM State Diagram for CRB Interface

End of informative comment

6.5.3.8 TPM CRB Buffer

6.5.3.8.1 Access Restrictions

Start of informative comment

The command/response buffer may be implemented as a single buffer or as individual buffers. The data buffer is specified as having a minimum size large enough to handle the largest implemented TPM command or response with the maximum number of authorization handles and a hash algorithm of SHA-256.

The command/response buffer can only be accessed initially at the base address. Subsequent accesses are required to go to sequential addresses. In the case where Software wants to reread a previous location, it must start at the base address.

End of informative comment

6.5.3.8.2 TPM_CRB_DATA_BUFFER_x Access Restrictions

1. On a write to the buffer:
 - a. If the initial transaction is to an address other than the base address for the buffer, a TPM MAY ignore the write.
 - b. If subsequent transactions are to non-sequential addresses, a TPM MAY ignore the write.
 - c. If the transaction is a size other than 1 byte or a power of 2 bytes, a TPM MAY ignore the write.
 - d. A TPM MAY invalidate any existing data within the buffer upon a write to the base address of the buffer.
 - e. A TPM SHALL replace invalidated data with data written upon a write to the base address of the buffer.
 - f. Following the write to the buffer and prior to a write to TPM_CRB_CTRL_START_x, a TPM MAY ignore read requests.
2. On a read from the buffer:
 - a. If the initial transaction is to an address other than the base address for the buffer, a TPM MAY ABORT the transaction, as defined in Section 6.5.1.1 Bus Aborts.
 - b. If subsequent transactions are to non-sequential addresses, a TPM MAY abort the transaction, as defined in Section 6.5.1.1 Bus Aborts.
 - c. Following the first transaction, if a subsequent read from the base address occurs, a TPM SHALL return the data at the base address.
 - d. On a write to the buffer, a TPM MAY ignore the data.
 - e. A TPM SHALL maintain the response in the buffer until receipt of a write of 1 to TPM_CRB_CTRL_REQ_x.goldle or TPM_CRB_CTRL_REQ_x.cmdReady.

6.5.3.9 CRB Interface State Transitions

Table 41 shows the changes in CRB Interface fields based on the command or action done to a TPM. Notice this is not a state transition table covering the states defined in 6.5.3.7 Interface Controls, rather a table describing how the status fields change based on initial condition and action taken. The following rules apply to Table 41.

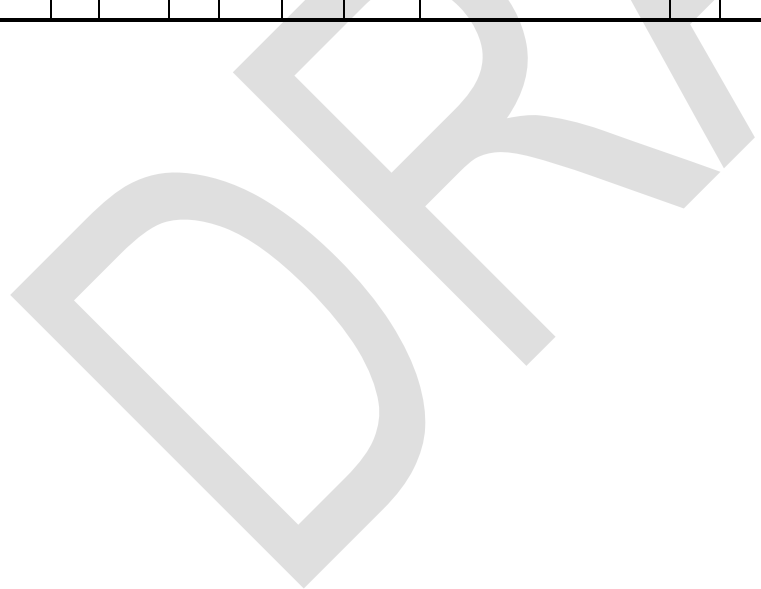
1. There MAY be intermediate status field states where a command has finished but TPM_CRB_CTRL_START_x field is not yet cleared. Software is expected to poll until the appropriate status field is set.
2. Table 41 applies only to CRB Interface field states where a locality has already been selected and no change in locality is performed.
3. The statements in the column labeled "Action Taken" are informative when shaded and are derived from normative statements contained within the definitions of the CRB Interface in Section 6.5.3 CRB Interface Requirements. If there is an inconsistency between the Action Taken column and those normative statements, the normative statements take precedence.
4. Normal transitions are highlighted in yellow and are indexed to the state transitions illustrated in Figure 4.
5. The following abbreviations are used in Table 41:

Label	Bit Definition
C/R	TPM_CRB_CTRL_REQ.cmdReady
gl	TPM_CRB_CTRL_REQ.goldle
I	TPM_CRB_CTRL_STS.tpmIdle
St	TPM_CRB_CTRL_STS.tpmSts
S	TPM_START
CC	TPM_COMMAND_CANCEL
—	No TPM access for the corresponding data element
X	Either 0 or 1. A TPM is allowed to maintain this field as either value for this state. Software must always be capable of managing the TPM, irrespective of the value

Table 41 — CRB Interface State Transitions

#	Present State of Control Area							Fields/Data Written					Next State & Result					Action Taken		
	State	Request		Status		S	CC	Request			S	CC	State (Row)	Request		Status		S	CC	Description
		C/R	gl	I	St			C/R	gl	Command Response Buffer				C/R	gl	I	St			
0	Init	-	-	-	-	-	-	X	X	-	X	X	Idle (1)	0	0	1	0	0	0	TPM Transitions from Init to Idle within TIMEOUT_B
1	Idle	0	0	1	0	0	0	1	0	-	X	X	Ready	0	0	0	0	0	0	TPM Transitions to Ready within TIMEOUT_C
2	Idle	0	0	1	0	0	0	0	1	-	X	X	Idle	0	0	1	0	0	0	TPM remains in Idle. TPM MAY ignore request.
3	Idle	0	0	1	0	0	0	X	X	-	1	0	Idle	0	0	1	0	0	0	TPM remains in Idle. TPM ignores request
4	Idle	0	0	1	0	0	0	X	X	-	0	1	Idle	0	0	1	0	0	0	No command to Cancel. TPM remains in Idle. TPM SHOULD ignore Cancel field.
5	Idle	0	0	1	0	0	0	X	X	Write access to buffer	0	0	Idle	0	0	1	0	0	0	TPM SHALL ignore any access to the C/R buffer
6	Idle	0	0	1	0	0	0	X	X	Read access to buffer	0	0	Idle	0	0	1	0	0	0	TPM SHALL ignore any access to the C/R buffer
7	Ready	0	0	0	0	0	0	1	0	-	0	0	Ready	0	0	0	0	0	0	TPM remains in Ready. TPM SHALL ignore request
8	Ready	0	0	0	0	0	0	0	1	-	0	0	Idle	0	0	1	0	0	0	Interface Reset. TPM goes to Idle state.
9	Ready	0	0	0	0	0	0	X	X	Write access to the buffer	0	0	Reception	0	0	0	0	0	0	TPM receives command
10	Ready	0	0	0	0	0	0	0	0	-	1	0	Ready	0	0	0	0	0	0	No data in the buffer, TPM remains in Ready. TPM SHALL ignore request
11	Ready	0	0	0	0	0	0	0	0	-	0	1	Ready	0	0	0	0	0	0	No command to cancel. TPM remains in Ready. TPM SHALL ignore request.
12	Ready	0	0	0	0	0	0	X	X	Read access to buffer	0	0	Ready	0	0	0	0	0	0	No response to read. TPM remains in Ready. TPM SHOULD ignore request.
13	Reception	0	0	0	0	0	0	0	0	-	1	0	Execution	0	0	0	0	1	0	TPM transitions to Execution state. TPM begins processing command.
14	Reception	0	0	0	0	0	0	0	0	-	0	1	Reception	0	0	0	0	0	0	TPM remains in Reception. TPM SHALL ignore request.
15	Reception	0	0	0	0	0	0	1	0	-	0	0	Ready	0	0	0	0	0	0	TPM transitions to Ready.
16	Reception	0	0	0	0	0	0	0	0	-	0	0	Reception	0	0	0	0	0	0	TPM remains in Reception.
17	Reception	0	0	0	0	0	0	0	0	Read access to buffer	0	0	Reception	0	0	0	0	0	0	TPM remains in Reception. TPM SHALL ignore request.
18	Reception	0	0	0	0	0	0	0	0	Write access to the buffer	0	0	Reception	0	0	0	0	0	0	TPM in Reception. TPM continues to receive data.
19	Reception	0	0	0	0	0	0	0	1	-	0	0	Idle	0	0	1	0	0	0	Abort Command Reception and return to Idle.
20	Execution	0	0	0	0	1	0	0	0	-	1	0	Execution	0	0	0	0	1	0	TPM remains in Execution state. TPM MAY ignore writes to Start field.
21	Execution	0	0	0	0	1	0	1	0	-	X	0	Execution	0	0	0	0	1	0	TPM remains in Execution state. TPM SHALL ignore request.
22	Execution	0	0	0	0	1	0	0	1	-	X	0	Execution	0	0	0	0	1	0	TPM remains in Execution state. TPM SHALL ignore request.

23	Execution	0	0	0	0	1	0	0	0	-	X	0	Completion	0	0	0	0	0	0	TPM clears Start field and transitions to Completion.
24	Execution	0	0	0	0	1	1	0	0		X	1	Completion	0	0	0	0	0	0	A TPM shall transition to Completion. TPM MAY Cancel command. TPM SHALL respond with Return code within TIMEOUT_B.
25	Execution	0	0	0	0	1	0	0	0	Write access to buffer	X	0	Execution	0	0	0	0	1	0	TPM remains in Execution state. TPM MAY ignore write requests
26	Execution	0	0	0	0	1	0	0	0	Read access to buffer	X	0	Execution	0	0	0	0	1	0	TPM remains in Execution state. TPM MAY ignore read requests
27	Completion	0	0	0	0	0	0	1	0	-	0	0	Completion	0	0	0	0	0	0	TPM remains in Completion. TPM SHALL ignore request.
27a	Completion	0	0	0	0	0	0	1	0	-	0	0	Ready	0	0	0	0	0	0	TPM SHALL transition to Ready if TPM_CAPCRB_INTF.CapCRBIdleBypass is set to 1.
27b	Completion	0	0	0	0	0	0	0	0	Write access to buffer	0	0	Reception	0	0	0	0	0	0	TPM receives command if TPM_CAP_CRB_INTF.CapCRBIdleBypass is set to 1.
28	Completion	0	0	0	0	0	0	0	1	-	0	0	Idle	0	0	1	0	0	0	TPM transitions to Idle, invalidates buffer.
29	Completion	0	0	0	0	0	0	0	0	-	1	0	Completion	0	0	0	0	0	0	TPM remains in Completion. TPM SHALL ignore request
30	Completion	0	0	0	0	0	0	0	0	-	0	1	Completion	0	0	0	0	0	0	TPM remains in Completion. TPM SHALL ignore request
31	Completion	0	0	0	0	0	0	0	0	Read access to buffer, sequential accesses	0	0	Completion	0	0	0	0	0	0	TPM returns data, remains in Completion and maintains data in buffer.
32	Completion	0	0	0	0	0	0	0	0	Read access to buffer, non-sequential accesses	0	0	Completion	0	0	0	0	0	0	Invalid access. TPM MAY ignore the transaction.
33	Completion	0	0	0	0	0	0	0	0	Read access, re-read from Base address	0	0	Completion	0	0	0	0	0	0	Retry. TPM resets internal byte counter and returns response from Base address.



6.6 Interrupts

Start of informative comment

As use of a TPM is non-preemptive (except for the special case of Seize) and a TPM is single threaded, there is no issue regarding sharing or colliding interrupts across localities. For example, if one locality starts a TPM operation, it cannot release a TPM to another locality until the pending TPM operation completes. However, if one locality (e.g., Locality 0) starts a long TPM operation, then turns control to another locality (e.g., Locality 2) before the long operation completes (and of course does not relinquish a TPM, which would cause a command abort), the second locality (e.g., Locality 2) would not know what the interrupt is for. This situation is outside the purview of this specification and negotiation of interrupt handling is done by Software.

When an event occurs that causes a TPM to signal an interrupt, a TPM must set the appropriate fields in a TPM_INT_STATUS_x or TPM_CRB_INT_STS_x register. If a TPM has not already sent an interrupt, the 0 to 1 transition of a field in a TPM_INT_STATUS_x or a TPM_CRB_INT_STS_x register must cause a TPM to assert the appropriate interrupt per the SIRQ or PIRQ protocol. The interrupt service routine will read a TPM_INT_STATUS_x or TPM_CRB_INT_STS_x registers to determine the cause and take appropriate action. When the interrupt has been serviced, the interrupt service routine must do an End-of-Interrupt (EOI) to the I/O APIC to re-arm a TPM's interrupt in the I/O APIC. Then the interrupt service routine must also send a TPM_EOI to a TPM to allow it to send new interrupts.

A TPM must not issue another interrupt until it has received its TPM_EOI message (see below), even if new events occur that should cause an interrupt. A TPM should set the appropriate field in TPM_INT_STATUS_x or TPM_CRB_INT_STS_x register, but the actual assertion of the interrupt will only occur after a TPM_EOI. If the interrupt handler detects multiple fields set, it may handle all the causes and clear multiple status fields. This means that an interrupt may be handled without causing a new interrupt.

A TPM_EOI to a TPM comprises writing a 1 to the field in a TPM_INT_STATUS_x or TPM_CRB_INT_STS_x register that corresponds to the type of interrupt just handled. Software may write multiple fields if it has handled multiple interrupts at one time.

The following informative sections provide further clarification. They should not change functionality.

If there are multiple fields set in the Interrupt register, and Software does not clear all the interrupts, then a TPM must issue another interrupt after it sees a TPM_EOI, which is a write to the Interrupt register.

The Software must not change the state of a TPM_INT_ENABLE_x.globalIntEnable or TPM_CRB_INT_ENABLE_x.globalIntEnable flag while an interrupt is active.

For example: if, after the write to the Interrupt register, there are fields still set, a TPM issues another interrupt. If Software writes all the fields of the Interrupt register, so that the register contains zero after the write, no new interrupt would be generated.

This covers the case that Software handles one interrupt at a time and then returns. It also covers the case that Software handles all the interrupts it knows about, so writes multiple fields into the Interrupt register, but a new interrupt is flagged between the time Software read the Interrupt register and the time it wrote a TPM_EOI.

Note 1:

Many commands respond immediately so during normal operation the driver, after sending a command, should poll a TPM for a response keeping the interrupts masked. If the driver determines that a TPM will not be able to respond immediately, it will stop polling a TPM and unmask the appropriate set of interrupts. If the driver does this, there is a possibility of a race condition between the time the interrupt is unmasked and the state being checked. Therefore, after unmasking the interrupt(s), the driver should poll a TPM one more time.

Note 2:

The interrupts defined in Section 6.6.1 FIFO Interrupts apply to both LPC and SPI interfaces. The interrupts defined in Section 6.6.2 CRB Interrupts support only the SPI interface. Interrupts for I2C are defined in sections 8.3.5.3 TPM_INT_ENABLE, 8.3.5.4 TPM_INT_STATUS, and 8.3.5.5 TPM_INT_CAPABILITY.

End of informative comment

1. A TPM SHALL set the appropriate field indicating the cause of the interrupt in a TPM_INT_STATUS_x or TPM_CRB_INT_STS_x register.
2. Once an interrupt is asserted, a TPM SHALL NOT assert another interrupt until it receives a TPM_EOI even if new events occur that should cause an interrupt.
3. A TPM has only one interrupt assigned to it, so interrupt settings for one locality SHALL be applied to all localities.

6.6.1 FIFO Interrupts**Start of informative comment**

The method for asserting interrupts uses the Serial-IRQ (SIRQ) protocol for interrupts. The protocol emulates the set of individual hardware signals using time division multiplexing between frames. An understanding of the SIRQ protocol is critical to a TPM implementer. The direct assertion of the SIRQ line does not signal an interrupt. The assertion of the interrupt is a combination of the change in the SIRQ signal during a time slot designated for that interrupt number. The state (level, edge, high, low) is expressed as the state of the SIRQ line during the assigned time slot over a series of frames.

A TPM must be designed to support assertion of any of the IRQ[0:15]. Certain platforms may not support certain IRQs being assigned to a TPM: therefore, a TPM must be capable of asserting any of the 16 possible IRQs. A TPM must not assert PIRQ[A:D] in the SIRQ stream.

There is a capability register that allows each platform to indicate to a TPM which interrupts the platform supports.

A TPM reports all schemes it supports in the Interrupt Capabilities register bits 3-6. The Software selects the scheme using the Interrupt Enable register bits 3-4. If a TPM supports only one scheme, bits 3 and 4 may be read only and return the value of the implemented scheme.

End of informative comment

1. A TPM SHALL support the following interrupts:
 - a. TPM_INT_STATUS_x.localityChangeIntOccured
 - b. TPM_INT_STATUS_x.dataAvailIntOccured
2. A TPM MAY support the following interrupts:
 - a. TPM_INT_STATUS_x.stsValidIntOccured
 - b. TPM_INT_STATUS_x.commandReadyIntOccured
3. A TPM SHALL support asserting any of the IRQ[0:15]. A TPM SHALL NOT assert PIRQ[A:D] in the SIRQ stream.
4. A TPM SHALL support low level interrupts, defined in Table 42, and MAY support the other interrupts. A TPM SHALL report all schemes it supports in the Interface Capabilities register.
5. If a TPM supports only one scheme, bits 3 and 4 MAY be read-only and return the value of the implemented scheme.
6. A TPM SHALL maintain interrupts as inactive during any change to a TPM_INT_ENABLE_x.globalIntEnable and while TPM_INT_ENABLE_x.globalIntEnable is 0.

6.6.1.1 FIFO Interrupt Enable

Table 42 — FIFO Interrupt Enable

Abbreviation:		TPM_INT_ENABLE_x		
General Description:		Enables specific interrupts and has the global enable. A TPM SHALL implement this register.		
Bit Descriptions:				
31	Read/Write	globalIntEnable	Default: 0	1 = Interrupts controlled by individual bits 0 = All interrupts disabled. cleared to 0 on reset.
30:8		Reserved	Reads always return 0	
7	Read/Write	commandReadyEnable	Default: 0	1 = Enabled 0 = Disabled
6:5		Reserved	Reads always return 0	
4:3		Reserved	Reads always return 01	
2	Read/Write	localityChangeIntEnable	Default: 0	1 = Enabled 0 = Disabled
1	Read/Write	stsValidIntEnable	Default: 0	1 = Enabled 0 = Disabled
0	Read/Write	dataAvailIntEnable	Default: 0	1 = Enabled 0 = Disabled

6.6.1.2 Interrupt Status

Table 43 — Interrupt Status

Abbreviation:		TPM_INT_STATUS_x		
General Description:		Shows which interrupt has occurred. A TPM SHALL implement this register.		
Bit Descriptions:				
31:8		Reserved	Default: 0	Reads always return 0
7	Read / Write 1	commandReadyIntOccured	Default: 0	When 1, indicates that a TPM_STS_x.commandReady field transitioned from 0 to 1. Writing a 1 to this field clears the interrupt. Writing a 0 to this field has no effect.
6:3		reserved	Default: 0	Reads always return 0
2	Read / Write 1	localityChangeIntOccured	Default: 0	When 1, indicates that the locality change interrupt occurred. This interrupt is caused whenever any locality moves from TPM_ACCESS_x.requestUse to TPM_ACCESS_x.activeLocality whenever this transition had been delayed due to another locality having TPM_ACCESS_x.activeLocality set. Note that if a TPM has no TPM_ACCESS_x.activeLocality set when TPM_ACCESS_x.requestUse is written, a TPM SHALL move directly from TPM_ACCESS_x.requestUse to TPM_ACCESS_x.activeLocality without causing the interrupt. Writing a 1 to this field clears the interrupt (i.e., a TPM_EOI). Writing a 0 to this field has no effect.
1	Read / Write 1	stsValidIntOccured	Default: 0	This interrupt indicates that a 0 to 1 transition on TPM_STS_x.stsValid. Writing a 1 to this field clears the interrupt. Writing a 0 to this field has no effect.
0	Read / Write 1	dataAvailIntOccured	Default: 0	This interrupt indicates that TPM_STS_x.dataAvail transitioned from a 0 to a 1. This 0 to 1 transition occurs when the command has been completed and there is a Response to be read. This transition SHALL only occur when both TPM_STS_x.dataAvail and TPM_STS_x.stsValid fields are 1. Writing a 1 to this field clears the interrupt. Writing a 0 to this field has no effect.

6.6.1.3 Interrupt Vector

Start of informative comment

A TPM_INT_VECTOR_x defined in Table 44 — Interrupt Vector only applies to the LPC interface.

End of informative comment

Table 44 — Interrupt Vector

Abbreviation:		TPM_INT_VECTOR_x		
General Description:		Contains the SIRQ value. A TPM SHALL implement this register.		
Bit Descriptions:				
7:4	Read Only	Reserved (0)	Default: 0	Read always return 0's; writes have no meaning.
3:0	Read / Write	sirqVec	Default: 0	A value of 0 means SIRQ is disabled and SIRQ is tristated. The SIRQ channel used by TPM can be from 1 to 15.

6.6.2 CRB Interrupts

Start of informative comment

During all interactions of the Software with a TPM there are several situations where Software must wait for a TPM to complete a requested action. Completion of a requested action will be indicated from a TPM to Software by a change of the corresponding status register change.

There are four state transitions that may provide a benefit for the overall system performance when being indicated to Software via an interrupt instead of polling:

Locality Change: Write 1 to TPM_LOC_CTRL_x.requestAccess => Wait for Locality x to be SET

Establishment Clear: Write to TPM_LOC_CTRL_x.resetEstablishmentBit => Wait for TPM_ESTABLISHMENT == 1

TPM Ready: Write 1 to TPM_CRB_CTRL_REQ_x.cmdReady => Wait for tpmlIdle == 0

Response Available: Write 1 to TPM_CRB_CTRL_x.Start => Wait for Start == 0

To allow for a flexible configuration and use of the interrupt it is necessary to provide the following fields:

Interrupt Enable: Allows configuration of a TPM; which interrupt source should be used.

Interrupt Status: Allows reading the source of an Interrupt asserted by a TPM.

End of informative comment

If a TPM implements interrupts on CRB, those interrupts SHALL be implemented as defined in Table 45 — CRB Interrupt Control.

6.6.2.1 CRB Interrupt Control Register

Table 45 — CRB Interrupt Control

Abbreviation:		TPM_CRB_INT_ENABLE_x		
General Description:		Used to Control CRB Interrupts		
Bit Descriptions:				
31	R/W	globalInterruptEnable	Default: 0	0 = All interrupts are disabled 1 = Interrupt enable is controlled by the individual bits in this register
30:4	R/O	Reserved	Default: 0	Reserved for future use
3	R/W	localityChangeIntEnable	Default: 0	0 = Disabled 1 = Enabled
2	R/W	establishmentClearIntEnable	Default: 0	0 = Disabled 1 = Enabled
1	R/W	cmdReadyIntEnable	Default: 0	0 = Disabled 1 = Enabled
0	R/W	startIntEnable	Default: 0	0 = Disabled 1 = Enabled

Table 46 — Interrupt Status

Abbreviation:			TPM_CRB_INT_STS_x	
General Description:			Shows which interrupt has occurred.	
Bit Descriptions:				
31:4	R/O	Reserved	Default: 0	Reserved for future use
3	Read/ Write 1	localityChangeInt	Default: 0	A 1 indicates that a locality change has occurred. This interrupt is caused whenever the value of bits 4:2 of a TPM_LOC_STATE register changes. Note: If a TPM has TPM_LOC_STATE_x.locAssigned == 0 before Request Use is set there will be no Interrupt because a TPM will make the transition immediately to the requesting locality Writing a 1 to this field clears the interrupt. Writing a 0 to this field has no effect.
2	Read/ Write 1	establishmentClearInt	Default: 0	A 1 indicates that the reset of a TPM_LOC_STATE_x.tpmEstablished field has been successfully executed after the corresponding TPM_LOC_CTRL_x.resetEstablishment request Writing a 1 to this field clears the interrupt. Writing a 0 to this field has no effect.
1	Read/ Write 1	cmdReadyInt	Default: 0	A 1 indicates that after a write of 1 to TPM_CTRL_REQ.cmdReady a TPM has successfully finished the transition to the Ready state (i.e. TPM_CRB_CTRL_STS_x.tpmIdle == 0) Writing a 1 to this field clears the interrupt. Writing a 0 to this field has no effect.
0	Read/ Write 1	startInt	Default: 0	A 1 indicates that a TPM has executed a command as requested by TPM_CTRL_Start_x = 0001 and the corresponding response is available for read-out (i.e. Start field has been cleared). This interrupt will also be triggered if the currently executed command will be cancelled via a Command Cancel. Writing a 1 to this field clears the interrupt. Writing a 0 to this field has no effect.

7 TPM Hardware

7.1 FIFO Interface Locality Usage per Register

Start of informative comment

Table 47 shows how a TPM responds to access to each of the interface registers based on which locality is active in the context of the FIFO interface. Table 47 only applies to Localities 0-3. It does not apply to Locality 4.

End of informative comment

1. If TPM_ACCESS_x.activeLocality setting changes when a command is executing, a TPM SHALL abort the currently executing command, as defined in Section 6.5.2.3.1 Command Aborts.

Table 47 — Register Behavior Based on Locality Setting for FIFO

TPM_ACCESS_x.activeLocality					
Set for this locality		Set for another locality		Not Set	
READ	WRITE	READ	WRITE	READ	WRITE
TPM_STS_x Registers					
TPM returns correct value	Fields updated	TPM returns FFh	TPM ignores the write	TPM returns FFh	TPM ignores the write
TPM_INT_ENABLE_x Registers					
TPM returns correct value	Field updated	TPM returns correct value	TPM ignores the write	TPM returns correct value	TPM ignores the write
TPM_INT_VECTOR_x Registers					
TPM returns correct value	Field updated	TPM returns correct value	TPM ignores the write	TPM returns correct value	TPM ignores the write
TPM_INT_STATUS_x Registers					
TPM returns correct value	Interrupt cleared	TPM returns correct value	TPM ignores the write	TPM returns correct value	TPM ignores the write
TPM_INTF_CAPABILITY_x Registers					
TPM returns correct value	Read-only register	TPM returns correct value	Read-only register	TPM returns correct value	Read-only register
TPM_ACCESS_x Registers					
TPM returns correct value	Fields updated	TPM returns correct value	Fields updated	TPM returns correct value	Fields updated
TPM_DATA_FIFO_x Registers					
TPM returns correct data	TPM accepts data and command	TPM returns FFh	TPM ignores the write	TPM returns FFh	TPM ignores the write
Configuration registers – 0F00h to 0FFFh					
TPM returns correct value	Fields updated	TPM returns correct value	TPM ignores the write	TPM returns correct value	TPM ignores the write

TPM_ACCESS_x.activeLocality					
Set for this locality		Set for another locality		Not Set	
READ	WRITE	READ	WRITE	READ	WRITE
TPM_HASH_START Register					
TPM returns FFh	TPM accepts command	TPM returns FFh	TPM ignores the write	TPM returns FFh	TPM accepts (and sets TPM_ACCESS_x.activeLocality for Locality 4)
TPM_HASH_DATA Register					
TPM returns FFh	TPM accepts data	TPM returns FFh	TPM ignores the write	TPM returns FFh	TPM ignores the write
TPM_HASH_END Register					
TPM returns FFh	TPM accepts command and clears TPM_ACCESS_x.activeLocality for Locality 4	TPM returns FFh	TPM ignores the write	TPM returns FFh	TPM ignores the write
TPM_DATA_CSUM Register					
TPM returns correct data	Read-only register	TPM returns FFh	Read-only register	TPM returns FFh	Read-only register
TPM_DATA_CSUM_ENABLE Register					
TPM returns correct value	Field updated	TPM returns correct value	Field updated	TPM returns correct value	Field updated

7.2 CRB Interface Locality Usage Per Register

Start of informative comment

Table 48 shows how a TPM responds to access to each of the interface registers based on which locality is active in the context of the CRB interface.

End of informative comment

1. If TPM_LOC_STATE_x.activeLocality setting changes when a command is executing, a TPM SHALL abort the currently executing command, as defined in Section 6.5.1.1 Bus Aborts.

Table 48 — Register Behavior Based on Locality Setting for CRB

TPM_LOC_STATE_x.activeLocality					
Set for this locality		Set for another locality		Not Set	
READ	WRITE	READ	WRITE	READ	WRITE
TPM_LOC_STATE_x Registers					
TPM returns correct value	Read-only register	TPM returns correct value	Read-only register	TPM returns correct value	Read-only register
TPM_LOC_CTRL_x Registers					
TPM returns 0	Field updated	TPM returns 0	Field updated	TPM returns 0	Field updated
TPM_LOC_STS_x Registers					
TPM returns correct value	Read-only register	TPM returns correct value	Read-only register	TPM returns correct value	Read-only register
TPM_CRB_INTF_ID_x Registers					
TPM returns correct value	Field updated	TPM returns correct value	Field updated	TPM returns correct value	Field updated
TPM_CRB_CTRL_x Registers					
TPM returns correct value	Fields updated	TPM returns FFh	TPM ignores the write	TPM returns FFh	TPM ignores the write
TPM_CRB_DATA_BUFFER_x Registers					
TPM returns response data	TPM accepts command or data	TPM returns FFh	TPM ignores the write	TPM returns FFh	TPM ignores the write
TPM_LOC_CTRL_4.TPM_HASH_START Field					
TPM returns 0	TPM accepts (and sets TPM_LOC_STATE_x.activeLocality to Locality 4)	TPM returns 0	TPM ignores the write	TPM returns 0	TPM accepts (and sets TPM_LOC_STATE_x.activeLocality to Locality 4)
TPM_LOC_CTRL_4.TPM_HASH_DATA Field					
TPM returns 0	TPM consumes data in command buffer	TPM returns 0	TPM ignores the write	TPM returns 0	TPM ignores the write
TPM_LOC_CTRL_4.TPM_HASH_END Field					
TPM returns 0	TPM finalizes, extends data and releases Locality	TPM returns 0	TPM ignores the write	TPM returns 0	TPM ignores the write
TPM_DATA_CSUM Register					
TPM returns correct data	Read-only register	TPM returns FFh	Read-only register	TPM returns FFh	Read-only register
TPM_DATA_CSUM_ENABLE Register					
TPM returns correct value	Field updated	TPM returns correct value	Field updated	TPM returns correct value	Field updated

7.3 TPM LPC Hardware Protocol

Start of informative comment

This specification addresses 2.0 compliant TPMs that make use of the LPC bus for connecting to the platform, as there are specific protocol requirements for TPM's using LPC. The definition of specific LPC requirements does not preclude the use of other interfaces on a 2.0 compliant TPM. The CRB interface is not supported by a TPM LPC Hardware Protocol.

NOTE: A TPM LPC Hardware Protocol is deprecated as of PTP 1.06 and may be removed in a future revision.

End of informative comment

1. If a TPM implements an LPC interface as the method of connecting to the chipset, it SHALL implement the LPC bus per the requirements of the LPC Interface Specification. A link may be found to the specification in Section 11 References.
2. If a TPM implements an LPC interface, a TPM MAY use the LPC CLKRUN# protocol for mobile platforms.
3. If a TPM implements an LPC interface, a TPM SHALL be designed such that the LPCPD# pin may be strapped high to disable the LPCPD# protocol.

7.3.1 LPC Locality Cycles for TPM Interface

Start of informative comment

This section only applies to TPM implementations using the LPC interface.

This specification defines two TPM specific LPC cycles, TPM-Write and TPM-Read, which were added for communication between the chipset and a TPM. This was done to prevent simple hardware attacks using a device on the LPC bus that decoded I/O or memory cycles using the previously defined START field. Cycles using the normal memory read/write or I/O read/write START field to the following ranges are not decoded by a TPM. On the LPC bus, apart from the START field, these cycles are identical to I/O cycles. These locality cycles are an additional indication to a TPM (beyond addressing) that the cycles are intended for a TPM as locality commands. These commands can only be generated by a trusted process, e.g., the chipset.

See Section 6.2.1 TPM Locality Levels for rules and restrictions on using the standard vs. Locality LPC cycles.

By definition, the Locality None level is lower than Locality 0.

End of informative comment

1. If a TPM supports Locality None and Locality None is the active locality, any TPM access request from Locality 0-4 is a higher locality priority. In this case, a TPM SHALL respond to Locality 0-4 register writes to TPM_ACCESS_x.requestUse and TPM_ACCESS_x.Seize per the requirements documented in Section 6.5.2.4 Access Register.

7.3.1.1 TPM-Write LPC Locality Cycle

start of informative comment

Table 49 shows a TPM-Write locality cycle format. It resembles the existing LPC I/O write.

If the CPU attempts to write more than 1 byte at a time to a TPM, the chipset must break this up into multiple cycles of 1 byte each to consecutive addresses.

End of informative comment

Table 49 — LPC Locality Cycle TPM-Write for Accessing a TPM

Field	Value for Bits [3:0]	Description
START	0101	Previously this was a reserved value. It is now allocated for TPM-Write and TPM-Read locality cycles.
CYCTYPE + DIR	0010	Same as used for standard LPC I/O Write
ADDR	See Description	Four nibbles. Same as the standard LPC I/O Write.
DATA-Low	DIGEST low nibble	
DATA-High	DIGEST high nibble	
TAR		Standard LPC TAR
SYNC		Standard SYNC field for an I/O Write
TAR		Standard LPC TAR

7.3.1.2 TPM-Read LPC Locality Cycle

Start of informative comment

Table 50 shows a TPM-Read locality cycle format. It resembles the existing LPC I/O read.

If the CPU attempts to read more than 1 byte at a time to a TPM, the chipset must break this up into a series of 1-byte reads to consecutive addresses.

End of informative comment

Table 50 — LPC Cycle TPM-Read for Accessing a TPM

Field	Value for Bits [3:0]	Description
START	0101	Previously this was a reserved value. It is now allocated for TPM-Write and TPM-Read.
CYCTYPE + DIR	0000	Same as used for standard LPC I/O Read
ADDR	See Description	Same as for TPM-Write
TAR		Standard LPC TAR
SYNC	Standard	Standard SYNC field for an I/O Read
DATA-Low	DIGEST low nibble	
DATA-High	DIGEST high nibble	
TAR		Standard LPC TAR

7.4 SPI Hardware Protocol

Start of informative comment

There were several goals that guided architecture of SPI hardware protocol and flow control for a TPM. These assumptions are as follows:

- A TPM must have a dedicated SPI ChipSelect# (CS#).
- Only the chipset is allowed to assert a TPM CS# signal. This means further that a TPM's CS# can only be connected to the south bridge.
- The SPI protocol should not break existing drivers or Software.
- A TPM Interface Specification 1.21 defines all registers as having a size of 4 bytes or less. This register size is maintained for compatibility with Software. This applies to the HASH_START/_DATA/_END sequence,

which may be generated by hardware because a TPM's data register is only 4 bytes. No additional registers are defined for registers that might be greater than 4 bytes. Future definitions of Software may support 8-byte or 64-byte data registers. The SPI flow control and protocol are defined to allow for 8-byte and 64-byte data transactions in case they are added later. This allows for future improvements in SPI throughput. An example would be a 64-byte data register at offset 0x80. The 4-byte data register is always implemented and available to Software to maintain backwards compatibility.

Note that a TPM-specific-accesses, defined for the LPC bus (TPM-Read and TPM-Write cycles) are not required for SPI.

End of informative comment

7.4.1 Clocking

Start of informative comment

The LPC interface has a free-running clock, but the clock defined by the SPI interface only runs when an SPI transaction occurs. A TPM, to maintain backwards compatibility, is not required to have an external free-running clock. TPM manufacturers may choose to support an external clock in their implementations. A TPM must, however, generate whatever clock source it needs to support internal command processing, as this command processing will likely take place when the SPI bus is idle, and the SPI clock is not running. Additionally, a TPM's timer/tick counter is based on this internal clock and does not require external clock support.

The SPI bus clock frequency may vary based on implementation and/or type of device attached. A TPM default clock frequency for PC Client platforms is defined to be 24MHz, but in future might be higher.

The SPI bus is a shared-bus architecture. As such, a PC OEM must take care to ensure compatibility between devices on a shared SPI bus. It is likely that there will be BIOS initiated accesses to an SPI ROM containing BIOS code on the same physical bus as an SPI TPM. The PC OEM has two options to deal with this case:

- 1) The platform and BIOS may be designed to start up at a frequency of 24MHz, as a TPM must support that clock frequency. The BIOS may increase the frequency for transactions to the BIOS ROM at a later point in POST.
- 2) The platform may be designed with a strap or other hardware method to force operation at a frequency, as the PC OEM may select a TPM with support for that frequency.

Note: The SPI bus may be shared. Therefore, when a TPM's CS# is not asserted, the SPI clock may be running at a faster frequency than a TPM supports. Since BIOS knows what TPM is attached to which south bridge, it will need to comprehend the frequency support for both components and will change the SPI clock frequency for a TPM segment while SPI is idle. There is no communication to a TPM of what the SPI clock frequency is. The frequency can change from command to command if the SPI bus is idle when the frequency changes.

Note: TPMs may be used in many types of PC Client platforms. In lower power environments, it is entirely likely that the SPI interface will run at a slower clock frequency, while in high performance environments, the interface will run at a higher frequency. This specification provides a range within which all TPMs will correctly function and allows for TPM vendors to differentiate their parts to allow for a variety of implementations. PC Client systems have multiple standard clock frequencies available which could be used as the source clocks for the SPI interface, such as 14.3MHz, 24MHz, 33MHz, and 66MHz. To enable the widest range of applications, TPM vendors are encouraged to support frequencies between 33MHz and 66MHz in addition to the required clock operating range to allow for higher performance applications.

End of informative comment

1. A TPM SHALL support an SPI clock frequency range of 10 - 24MHz.
2. A TPM MAY support running at lower frequencies.
3. A TPM SHOULD support higher frequencies.

7.4.2 Electrical Specification

Start of informative comment

The SPI interface does not have an industry standard for electrical characteristics that can be referenced for TPM implementations as was done for LPC. This section describes the normative requirements for a TPM as defined at a TPM pins. This does not describe the requirements for the south bridge.

End of informative comment

1. A TPM SHALL support a supply and I/O voltage of 1.8V or 3.3V.
2. A TPM MAY support supply and I/O voltages of both 1.8 and 3.3V.
3. A TPM MAY support other supply and I/O voltages.
4. A TPM SHALL comply with the electrical specifications in the tables below.

NOTE: For the electrical specifications in Tables 45-47, the timing characteristics are defined only for the specified clock operating range. For other clock frequencies, the timing characteristics are implementation specific.

Table 51 — DC Specifications for 1.8V Supply Voltage

Parameter	Conditions	Min	Max	Units
Vcc power supply		1.65	1.95	V
VIH	Vcc = 1.65V – 1.95V	0.7 * Vcc	0.3 + Vcc	V
VIL	Vcc = 1.65V – 1.95V	-0.3	0.3 * Vcc	V
VOH	Vcc = 1.65V – 1.95V	0.9 * Vcc		Iout = -100μA
VOL	Vcc = 1.65V – 1.95V		0.1 * Vcc	1.5 mA

Table 52 — DC Specifications for 3.3V Supply Voltage

Parameter	Conditions	Min	Max	Units
Vcc power supply		3.0	3.6	V
VIH	Vcc= 3.0V – 3.6V	0.7*Vcc	0.5+VCC	V
VIL	Vcc= 3.0V – 3.6V	-0.5V	0.3*VCC	V
VOH	Vcc = 3.0V – 3.60V	0.9 * Vcc		Iout = -100μA
VOL	Vcc = 3.0V – 3.60V		0.1 * Vcc	1.5 mA

Table 53 — AC Electrical Specifications

Parameter	Description	Conditions	Min	Max	Units
	Clock minimum operating range		10	24	MHz
t_{CLKf}	Clock Period	Rising Edge to Rising Edge	$1/f_{CLK}-5\%$	$1/f_{CLK}+5\%$	ns
t_{CLKr}	Nominal Clock Period	Nominal Clock Period	$1/f_{CLK}$		ns
t_{CLKL}	Clock Low Time	Clock Low Time (see Figure 6)	$0.45t_{CLKr}$	-	ns
t_{CLKH}	Clock High Time	Clock High Time (see Figure 6)	$0.45*t_{CLKr}$	-	ns
$t_{CLKslew}$	Clock Slew Rate	$0.2*V_{CC} - 0.6*V_{CC}$	1	4	V/ns
t_{CS}	CS# High Time	Rising Edge to Falling Edge	50		ns
t_{CSS}	CS# Setup to clock	CS Setup time	5		ns
t_{CSH}	CS# Hold to clock	CS Hold time	5		ns
t_{SU}	MOSI Setup to clock	Data Setup time	2		ns
t_H	MOSI Hold to clock	Data Hold time	3		ns
t_{HO}	Clock to MISO valid	Output Hold time	0		ns
t_{vmin}	Output valid from clock falling edge minimum	Output Valid Min	0		ns
t_{vmax}	Output valid from clock falling edge maximum	Output Valid Max		$0.7* t_{CLKL}$	ns
	TPM SPI Pin Capacitance			10	pF

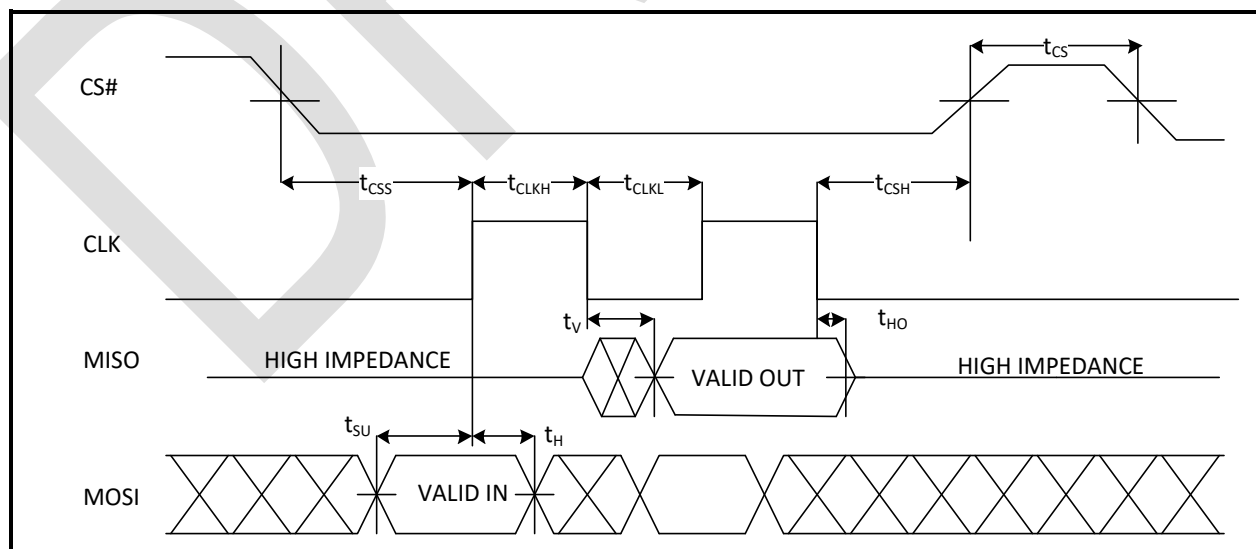


Figure 5 — Timing Diagram

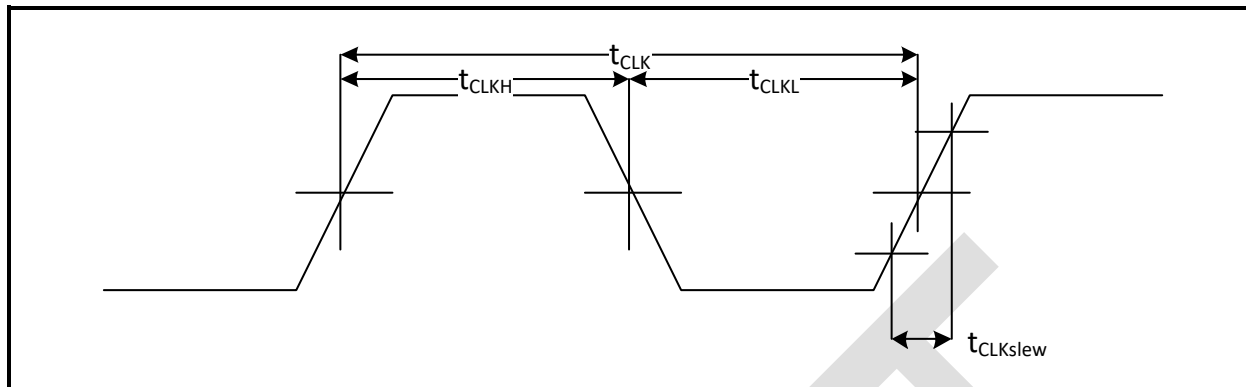


Figure 6 — Clock Timing Diagram

7.4.3 SPI Interrupts

Start of informative comment

TPMs compliant with the LPC specification support a serial interrupt, SIRQ. The SPI specification does not have an analog to SIRQ. This specification defines a parallel interrupt which functions in a manner resembling that of a PCI device's INTx#. The implementation of the pin is active low and open collector such that it is sharable. TPMs might continue to support an SIRQ signal to allow for a common design supporting either interface, but they are not required to do so.

End of informative comment

1. A TPM SHALL implement a PIRQ# pin.
2. PIRQ# SHALL be active low.
3. PIRQ# SHALL be open collector.
4. The PIRQ# pin SHALL be 3.3V tolerant.
5. A TPM MAY implement an SIRQ pin in addition to PIRQ#
6. A TPM MUST NOT implement an internal pull-up resistor on PIRQ#.

7.4.4 Legacy I/O

Start of informative comment

Previous versions of this specification contained support for legacy I/O cycles and addresses supported by TPM1.1b. For the SPI interface, this version of this specification deprecates all support for legacy accesses to a TPM, including the legacy I/O range. If a TPM vendor chooses to continue to support the LPC interface, they may continue to support Legacy addressing on the LPC interface only. It is expected that south bridges supporting SPI as defined in this specification will block any I/O cycles to TPMs connected via an SPI bus. This will be enforced in hardware. TPMs compliant with this specification do not need to implement the legacy IOW/IOR access mechanism. The south bridge will route the entire address range from xxD4_0000h through xxD4_4FFFh to a TPM over SPI. This allows a TPM to add new registers and maintain compatibility with the south bridge.

End of informative comment

7.4.5 Flow Control

Start of informative comment

The SPI interface does not define a flow control mechanism. A TPM, as defined, requires flow control to allow for varying sized data transfers.

This specification defines a method of flow control that operates on a transaction basis. On the LPC interface, transfers occur on a byte-by-byte basis, regardless of the transaction size. On SPI, transfers may occur in 4-, 8-, 32- or 64-byte chunks.

The method of flow control specified in this section allows a TPM to insert a wait state to hold off the transfer. Additionally, each transaction will provide, as part of the packet preceding the address, a transaction size. This allows a TPM vendor to implement more advanced mechanisms of flow control. For example, if the south bridge initiates a write of 32 bytes to a TPM_XDATA_FIFO or a TPM_CRB_DATA_BUFFER, a TPM can read the size of the transfer and, if it does not have 32 bytes of open space in its buffer, it would insert a wait state. Alternatively, a TPM could accept the transaction if it had a 64-byte buffer with only 8 bytes of data present. A TPM vendor isn't required to verify transaction sizes before resorting to the flow control mechanism specified here. A TPM may choose to ignore the flow control method and choose to insert wait states, as defined here, if a TPM's buffer is not currently empty. This will have a performance impact on larger transaction sizes but should pose no issue with 4-byte or 8-byte transfers. Byte level flow control was not considered, as the overhead of allowing flow control between each byte is too high with almost no benefit.

To allow flexibility for larger size transactions in the future, south bridges are likely to have limited, if any, HW checking on the size of accesses to a TPM address space. If the south bridge (SB) receives a transaction for any size from 1 byte to 64 bytes that doesn't cross a 64-byte boundary, it may choose to accept and issue that transaction to a TPM on SPI as received. A TPM, if it doesn't insert a wait state at the designated point must accept the transaction, if the transaction doesn't cross a register boundary. If the transaction crosses a register boundary, a TPM may choose to accept all the data and discard the data that exceeds the size limit for that register if so doing does not cause a change to the state of any adjacent register. The flow control specified in this specification defines a transaction structure for SPI consisting of 1 byte of command (including direction of transfer and transaction size), 3B of address, followed by the transaction data (either write data from the south bridge to a TPM or read data from a TPM to the south bridge). A TPM may insert wait states following receipt of the address. The south bridge will monitor the MISO line on the rising edge of the clock in the window following transmission of the last bit of the address. A TPM, to insert a wait state, drives the MISO line low (0) on the falling edge of the previous clock (clock in which the last bit of address is driven by the south bridge). A TPM would continue to drive MISO line in 8-bit increments until it is ready to receive or transmit data. The south bridge polls the MISO line every 8 clocks until it sees a 1, then it either starts to transmit data or expects to receive data on the next falling clock edge.

Note: For the purposes of defining the flow control, on SPI the MOSI or MISO signal is driven by the owner on the clock's falling edge and captured by the receiver on the clock's rising edge.

For a read, the command and address are driven on MOSI and a TPM responds with data on MISO. With no wait states, a TPM would drive data on the next falling clock edge following receipt of the last address bit on the rising edge of the clock. This is illustrated in Figure 7 below. The SB monitors the MISO pin in the same clock window that A[0] (the last address bit) is valid. If MISO is captured high on the rising edge of the clock, then the SB will continue to write or read data on the following clock edges. This aligns with standard SPI protocol where there are no wait states. In Figure 7 — Example Read transaction with a WAIT state and Figure 8, if a TPM drives a 1 in the A[0] window, or in any subsequent wait state window, then MISO is no longer used for wait states, in which case MISO is either providing valid data for reads or is a don't care for writes.

No other flow control is allowed. Once data starts coming from a TPM, it must provide the entire transaction's worth of data, which can be from 1 to 64 bytes, depending on a TPM's supported transaction size. In this example, if the south bridge had latched a 1 on the rising clock in the wait state window, then it would start latching in data starting on the next rising edge, which is the normal behavior without wait states. A TPM may insert any number of wait states that it needs.

Since the wait state is defined as 0 on MISO, if there is no TPM present at all, then the SPI interface design has a weak pull-up on MISO. If a TPM is not present, a pull-up on MISO means that the SB controller sees a 1 and will latch in 0xFF on the read. This follows standard master abort behavior of 0xFF for read data and matches the behavior when a TPM was on LPC.

For writes, the mechanism is similar. If MISO is 0 to request a wait state, then the data driven during that byte is not valid and a TPM will drop the data. If there is a wait state, the master will drive the first byte of the transaction until the slave stops requesting wait states. The SB will sample MISO on the last data bit of the byte (multiples of 8 clocks after the first wait state window). Again, a TPM must hold MISO as 0 for 8 clocks each time it requests a wait state. If MISO is 1, this indicates that a TPM is ready for the entire write, accepts the first byte which the SB has sent in the same clock, and the SB will then drive the 2nd and subsequent bytes on the following clocks. Once the data starts

transmitting, the entire write data will be sent with no further flow control. See Figure 8 — Example of WRITE transaction with Wait state for an example of a write transaction with wait states inserted.

Wait states are byte based. For reads where MISO is used to return data to the master, the SB will start sampling at byte[-1], which is the window when A[7:0] is transmitted. If the last bit of this window is 0, then a TPM is requesting a wait state and the SB continues reading MISO for 8 clocks (a byte's worth) and will look at the last bit to determine if there should be another wait state or not. From the SB standpoint, it is simply reading a byte and processing it as a byte. The last bit of that byte determines what to do next. If the bit is 1, then the SB knows to start sending the valid data on writes or receiving data for reads. One option on writes is to send data byte 0 over and over until there is no wait state, and then move to byte 1, etc. For reads, each byte can be sampled, and when the last bit is 1, start moving the next byte into the data buffer and increment the byte count received.

Note: A TPM interface was architected knowing that some of the actual command processing could take seconds to complete. Therefore, registers were provided so Software can poll to determine when it should read the actual results of the command. Software should never attempt to read the DATA FIFO without verifying that a TPM_STS_x.commandReady and TPM_STS_x.dataAvail fields are set. If Software does, a TPM will return FF's as described in Section 6.5.1.1 Bus Aborts. For writes, a TPM is required to insert wait states if Software attempts to write data without waiting for a TPM to transition to the Ready state. The HW will allow flow control until a TPM is ready to provide the data, which could be as long as the applicable timeout. On the other hand, the registers used for the Software control must not have excessively long delays or the system performance would be impacted. There are some registers for which wait states would present a problem in the overall operation of the system. A TPM is only allowed 1 wait state to decode the address before returning the contents of the register. The FIFO registers with this restriction are:

- 1) ACCESS (0x0), once the requirements defined in Section 7.6 Reset Timing are satisfied.
- 2) INT_ENABLE (0x8)
- 3) INT_VECTOR (0xC)
- 4) INT_STATUS (0x10)
- 5) INTF_CAPABILITY (0x14)
- 6) STS (0x18), after the register contains a valid logical level as defined in Section 6.5.2.5 Status Register
- 7) HASH_START (0x28)
- 8) DID_VID (0xF00)
- 9) RID (0xF04)

There are no CRB registers with this restriction.

Note: when inserting wait states on the bus, if that SPI segment is used by other devices, then they will be stalled until a TPM completes the transaction. Adding wait states slows down the system, so should be used sparingly.

Example of a read and write transaction with WAIT state are shown below.

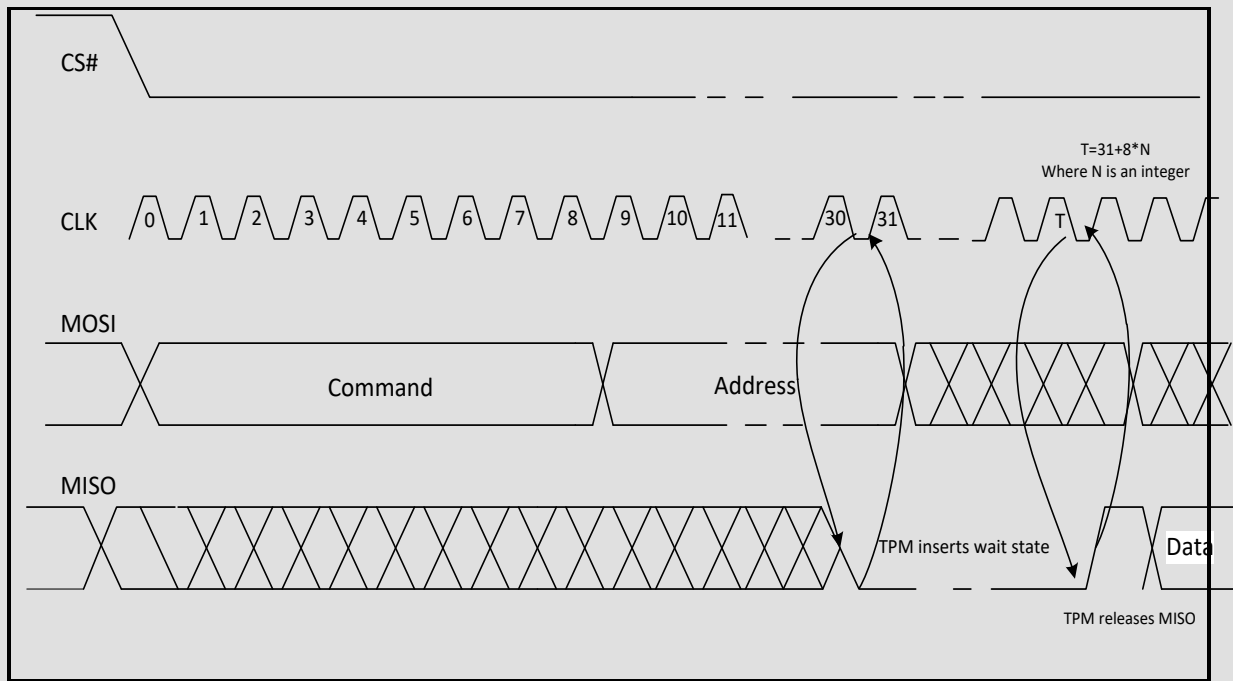


Figure 7 — Example Read transaction with a WAIT state

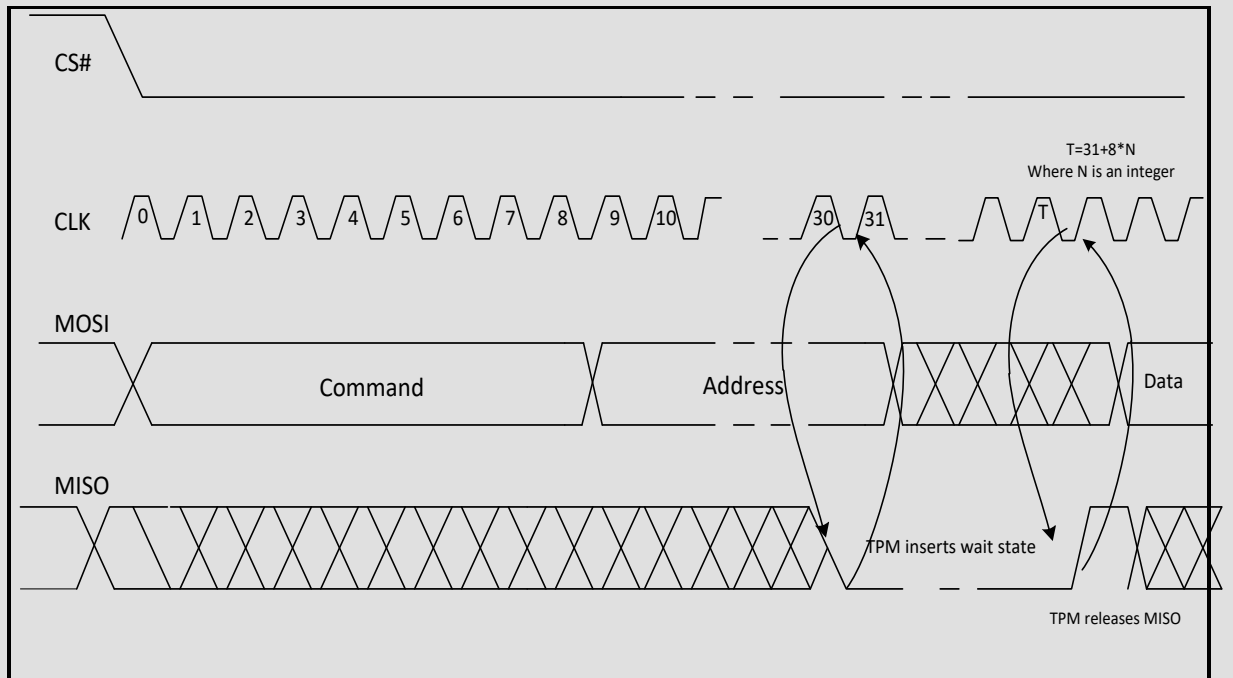


Figure 8 — Example of WRITE transaction with Wait state

End of informative comment

1. The wait state window is defined to start on the rising edge of the clock on the transmission of the last bit of address to the rising edge of the subsequent clock.

2. The wait state window repeats every 8 clocks in the same transaction until a TPM no longer inserts wait states.
3. A TPM SHALL drive MISO low (0) on the falling edge of the clock in the wait state window to signal insertion of a wait state.
4. A TPM SHALL continue to drive MISO low (0) on the falling edge of the clock in subsequent wait state windows for the same transaction until it is ready to receive (Read) or transmit (Write) all the data for that transaction.
5. A TPM SHALL drive MISO high (1) on the falling edge of the clock in the wait state window to signal no further wait states will be inserted.
6. A TPM SHALL drive MISO high (1) on the falling edge of the clock in the first wait state window if it does not intend to insert a wait state.
7. A TPM SHALL NOT insert more than 1 wait state on a Read cycle to the following registers:
 - a. TPM_ACCESS_x, once the requirements defined in Section 7.6 Reset Timing are satisfied.
 - b. TPM_STS_x, after the register contains a valid logical level as defined in Section 6.5.2.5 Status Register
 - c. TPM_INTF_CAPABILITY
 - d. TPM_INT_ENABLE
 - e. TPM_INT_STATUS
 - f. TPM_INT_VECTOR
 - g. TPM_DID
 - h. TPM_VID
8. A TPM MAY insert wait states for accesses to a TPM_HASH_x register, but SHALL NOT exceed TIMEOUT_B.

7.4.6 SPI Bit Protocol

Start of informative comment

The bit protocol defined in this section provides for transactions to follow the following rules:

- 1) Data is transferred most significant bit (msb) first, least significant byte (LSB) first
- 2) Address and command are transferred msb first for the entire field, e.g., the 24-bit address is transferred by sending A23 first, then A22 all the way to A0.
- 3) Master and slave both drive data on the falling edge of the SPI clock.
- 4) Master and slave both sample data on the rising edge of the SPI clock.
- 5) The address presented to a TPM on the SPI bus will always be a 24-bit address that is offset from the upper octet. The chipset will decide the full address and if the cycle is in the xxD4_xxxxh range, it will assert a TPM's CS# pin.
- 6) Only SPI mode 0 is supported (CPHA=0, CPOL=0).

The bit order defined below is transmitted on the wire starting from the bottom of the table, ending with the top of the table. The first bit in the protocol is the read/write bit, allowing a TPM to determine what type of transaction follows. The last bit is the least significant bit (lsb) of the most significant byte (MSB) of the data packet. As described in Section 7.4.5 Flow Control, it is legal to transmit any number of bytes of data from 1 byte to 64 bytes. Zero-length transactions are not supported or allowed.

There is no status byte built into the protocol. The existing methods using TPM_STS_x.burstCount and TPM_STS_x.Expect govern transfer failures, as defined in Section 6.5.2.5 Status Register. If a TPM has not received all of the bytes of the transaction, it will set TPM_STS_x.Expect to a 1, to signal to the chipset that it still expects data. On a read, if the chipset does not receive the required number of bytes (or any bytes), the chipset may issue a retry, or it may send all FF's to the driver, signaling a failure.

The SPI interface has evolved to support double data rate transactions. A TPM does not support double data rate transfers.

End of informative comment

1. A TPM SHALL support the bit protocol defined in Table 54.
2. A TPM SHALL drive read data on the falling edge of the clock.
3. A TPM SHALL sample write data on the rising edge of the clock.
4. A TPM SHALL decode transactions sent to offset 0xD4_xxxx when its CS# is asserted.

Table 54 — SPI Bit Protocol

Bit Transfer Order on MISO/MOSI	BYTE on MISO/MOSI	Usage	Notes
	67 for 64B xactions 11 for 8B xactions	future use for larger register sizes	
57-63 – last bits on wire	7	Data[30:24]	
56		Data[31]	msb of 4 th LSB
49-55	6	Data[22:16]	
48		Data[23]	msb of 3 rd LSB
41-47	5	Data[14:8]	
40		Data[15]	msb of 2 nd LSB
33-39	4	Data[6:0]	
32		Data[7]	msb of LSB
Optional flow control can be done in this window. See the Section 7.4.5 Flow Control for details. This is the only place in the bit transfers where flow control can be done.			
31	3	Addr[0]	lsb of address
9-30	1-3	Addr[22] down to Addr[1]	
8	1	Addr[23]	msb of address
2-7	0	bits[5:0] Size of transfer where bit[5] of this field is the 3 rd bit transferred on the wire, and bit [0] is the 8 th bit on the wire. This field is a 0's-based count of the bytes. Any byte count from 1 to 64 is legal.	Bit [5:0] decode '11_1111' = 64 bytes ' etc. for 63 down to 6 bytes '00_0100' = 5 bytes '00_0011' = 4 bytes '00_0010' = 3 bytes '00_0001' = 2 bytes '00_0000' = 1 byte
1		rsvd; bit[6]	
0 – first bit on wire		Byte0, bit[7] Read/Write	1=read, 0 = write

7.5 TPM Byte Ordering**Start of informative**

A TPM Interface Specification contains definitions for TPM registers which have multi-byte address ranges. Data transmitted on the interface to these registers is transferred from the lowest address or least significant byte (LSB at byte offset 0) to the highest address or most significant byte. For more information on the addressing and address

decode of these registers, see Sections 6.3 TPM Register Space, 6.5.2.5 Status Register, and 6.5.2.6 Data FIFO Register.

A TPM Library Specification [7] defines command structures which have multi-byte fields, which are defined as follows: Integer values are expressed as an array of one or more bytes. The byte at offset zero within the array is the most significant byte of the integer, referred to within this section as big-endian. For example, a field designation of UINT-32 is a byte array of 4 bytes with the most significant byte at byte offset 0. These commands are transmitted on a TPM interface as the payload of a TPM register access.

A TPM Interface does not ensure or validate the byte ordering of the payload. It is the responsibility of a TPM Software, typically a TPM driver in conjunction with the TSS, to correctly marshal the command payload for any write to a TPM register.

To write to a multi-byte register, e.g., a TPM_DATA_FIFO, a TPM must receive the least significant byte first, as defined in Section 6.3.1 TPM Register Space Decode. The payload of that transfer will contain the command, which may include multi-byte arrays, having their MSB at offset 0.

The driver is required to handle the byte ordering of TPM command fields versus the byte ordering of a TPM registers, the LPC bridge, and LPC bus. Software may do Double Word (DW) (4 bytes) or Word (2 bytes) or Byte accesses to a TPM. Standard PC platforms will have bridges that translate these 4-byte or 2-byte accesses into single-byte accesses on LPC. PC platforms will always break up the request so that they send the least significant address of that request first on the LPC bus.

As an example, a TPM data structure TPM_PROTOCOL_ID is defined in a TPM Library Specification [7] as having a value of 0x0005 for PID_OWNER, where 0x00 is the MSB in the array. To ensure a TPM receives the correct command payload, this structure must be sent to a TPM with 0x00 as the first byte and 0x05 as the second byte. However, the bridge between the CPU and a TPM sends the LSB first and therefore, a CPU write of 0x0005 would send the 0x05 to a TPM first, then the 0x00 which is not the correct sequence for the TCG_PROTOCOL_ID for PID_OWNER. The driver could perform two 1-byte accesses to a TPM, the first write would be with data 0x00, and the second write would be with data 0x05. Or Software could do a Word access and send 0x0500, which would result in the bridge issuing the 0x00 cycle first on the LPC bus followed by an LPC cycle of 0x05.

End of informative comment

7.6 Reset Timing

Start of informative comment

The operation of the platform's CRTM likely occurs during a very time-sensitive period. Because of this, strict requirements are necessary for a TPM's reset timing. During this time, the platform's CRTM may need to make decisions based on the presence or absence of a TPM's response that affect the rest of the platform's boot cycle. This requires that any return from TPM_ACCESS_x register be valid regardless of the timing – a TPM must not be allowed to return anything but a valid response from this register.

Note: A TPM may need to perform maintenance or recovery of NV in cases such as an unexpected power loss during a field upgrade, following an unexpected loss of power, or in the case of recovering from NV corruption. It may take longer for a TPM to be ready to receive a command.

While a TPM_ACCESS_x register is the most critical, the availability of the other registers is important for performance reasons.

This section contains the timing requirements for a TPM's registers. The normative requirements describing the relationship between the individual fields within a register are contained in Sections 6.5.2.4 Access Register, 6.5.2.5 Status Register, and 6.5.3 CRB Interface Requirements.

End of informative comment

1. Within 500 microseconds of the completion of _TPM_INIT:
 - a. For FIFO interface, all fields within all TPM_ACCESS_x registers SHALL be a valid logical level as indicated by a TPMRegValidSts field being set to a 0 or a1.

- b. For CRB interface, all fields within TPM_LOC_STATE_X register SHALL be a valid logical level as indicated by a TPMRegValidSts field being set to a 0 or a 1.
2. For the FIFO interface, all fields within the access register and all other registers SHALL return with the state of all their fields valid (i.e. TPM_ACCESS_x.tpmRegValidSts is set to 1) and a TPM SHALL be ready to receive a command (see Sections 6.5.1.4 Interface Timeouts and 6.5.1.5 _TPM_INIT and Reset):
 - a. For TPMs that do not comply with FIPS 140 Self-Test requirements, The TPM SHALL return the response within TIMEOUT_D, from the completion of _TPM_INIT.
 - b. For TPMs that do comply with FIPS 140-2 SelfTest requirements, the TPM SHALL return the register contents within 50ms from the completion of _TPM_INIT.
 - c. For TPMs that comply with FIPS 140-3 SelfTest requirements, the TPM SHALL return the response within 200ms of _TPM_INIT
 - d. For NV maintenance and recovery, the TPM SHALL return the response within 1 second of _TPM_INIT.
3. For CRB interface, all fields within a TPM_LOC_STATE_X, TPM_LOC_CTRL_X, TPM_LOC_STS_X, and TPM_CRB_INTF_ID_X shall be valid and a TPM SHALL be ready to receive a command (see Section 6.5.1.4 Interface Timeouts):
 - a. For TPMs that do not comply with FIPS 140-2 Self-Test requirements, the TPM SHALL be ready within TIMEOUT_D, from the completion of _TPM_INIT.
 - b. For TPMs that do comply with FIPS 140-2 SelfTest requirements, the TPM SHALL return the register contents within 50ms from the completion of _TPM_INIT.
 - c. For TPMs that comply with FIPS 140-3 SelfTest requirements, the TPM SHALL return the register contents within 200ms of _TPM_INIT
 - d. For NV maintenance and recovery, the TPM shall return the register contents within 1 second of _TPM_INIT.

8 I2C Interface Definition

Start of informative comment

This section defines the registers and hardware protocol for a PC Client TPM implemented with an I2C interface. The register definition is different from a FIFO or CRB interface as the I2C addressing scheme prevents 1:1 mapping of addresses. If a TPM vendor implements I2C, the requirements in this section are needed to comply with this specification.

End of informative comment

8.1 TPM I2C Interface Requirements

8.1.1 Bus speed

Start of informative comment

The speed of the data transfers is influenced by the participating components as well as by the clock frequency used. I2C provides options for several bus speeds, standard mode (Sm, up to 100 kbs), fast mode (Fm, up to 400 kbs), fast mode plus (Fm+, up to 1 Mbits/s), high speed mode (Hs-mode, up to 3.4 Mbits/s) and ultra-fast mode (UFm, up to 5 Mbits/s). To allow a wide usage without enhanced HW features, standard mode and fast mode are a good choice with suitable data transfer rates. As fast mode devices can communicate with standard mode devices, fast mode has been chosen.

End of informative comment

1. An I2C-TPM compliant to this specification SHALL be able to operate at Fast mode (Fm).
2. Higher speeds are allowed and SHALL be indicated via the Interface Capability register (see Section 8.3.5.10 of this document).

8.1.2 I2C Device address

Start of informative comment

Each device on the I2C bus needs a unique slave address. I2C offers two options, 7-bit slave address is mandatory and 10-bit slave address is optional. Because 10-bit addressing requires 2 bytes of overhead, 7-bit addressing (1 byte overhead) has been chosen to keep the overhead small.

The default 7-bit I2C device address is 0x2E, the 8th bit indicates the data direction. Therefore, the first byte after the START condition will be 0x5D for an I2C read request and 0x5C for an I2C write transmission.

End of informative comment

1. An I2C-TPM compliant to this specification SHALL support one 7-bit I2C device address.
2. Default address is 0x2E.
3. An I2C-TPM compliant to this specification MAY support reconfiguration of the I2C device address.
 - a. If supported the reconfiguration SHOULD follow the mechanism defined in Section 8.3.5.15 TPM_I2C_DEVICE_ADDRESS.
 - b. An I2C-TPM MAY implement a vendor defined mechanism for the reconfiguration of the I2C device address.

8.1.3 Fast turnaround

Start of informative comment

I2C offers two options for write / read cycles, one is to have two separate frames for write and read, and the other is to combine two frames using the repeated start condition. The second option allows a slightly higher throughput. Additionally, the repeated start mechanism avoids an allocation of the bus by any other device.

End of informative comment

1. An I2C-TPM compliant to this specification SHOULD support the repeated start condition (Sr) for I2C read after I2C write.

8.1.4 Data rate synchronization

Start of informative comment

I2C is a synchronous bus and the operating speed of the I2C-TPM may sometimes require that the communication speed of the master is throttled down so that the I2C-TPM has sufficient time to store data or to provide correct response data. To allow the synchronization between the I2C-TPM and the bus master it is required that the bus master supports the clock stretching mechanism.

End of informative comment

1. If an I2C-TPM needs to synchronize the data rate on the bus it SHALL use clock stretching.

8.1.5 Supply voltage

Start of informative comment

Besides the supply voltages required by this specification the I2C-TPM may also support other supply voltages.

End of informative comment

1. An I2C-TPM SHALL support a supply and I/O voltage of 1.8V or 3.3V.
2. An I2C-TPM MAY support supply and I/O voltages of both 1.8 and 3.3V.
3. An I2C-TPM MAY support other supply and I/O voltages.

8.1.6 Pull-up resistors

Start of informative comment

I2C needs pull-up resistors to allow the implementation of the wired-AND function. As I2C is critical with respect to the rise time of the bus signals the configuration of the pull-up resistors depends mainly on the bus length and on the number of devices connected to the bus. To allow sufficient flexibility regarding the configuration of the pull-up resistors, external resistors will be implemented on the platform.

End of informative comment

1. An I2C-TPM compliant to this specification SHALL NOT have internal pull-up resistors on the SDA and SCL pins.

8.1.7 Host interrupt

Start of informative comment

An I2C-TPM may need a considerable time to process certain requests especially for commands with cryptographic operations. Polling the I2C-TPM during command processing for response availability would create a certain bus load on the one hand and would also create a high system load. Therefore, it is better that the I2C-TPM informs the host once the command processing has been finished and the response is available. Furthermore, for some other events the host may desire notification regarding such events via an interrupt mechanism rather than polling for such an event.

End of informative comment

1. An I2C-TPM compliant to this specification SHALL support a host interrupt (PIRQ#) for signaling of certain events (e.g., response availability).

8.1.8 Availability after reset

Start of informative comment

At power-on after reset the I2C-TPM needs some time to perform the initialization of the device and the initial self-test. Therefore, it is necessary to allow the I2C-TPM a certain amount of time to perform such operations before it can handle communication requests.

End of informative comment

Following the completion of `_TPM_INIT`:

1. For an I2C-TPM, all fields within the access register and all other registers SHALL return with the state of all their fields valid (i.e. TPM_ACCESS_x.tpmRegValidSts is set to 1) and a TPM SHALL be ready to receive a command (see Sections 6.5.1.4 Interface Timeouts and 6.5.1.5 _TPM_INIT and Reset):
 - a. For TPMs that do not comply with FIPS 140 Self-Test requirements, the TPM SHALL return the register contents within TIMEOUT_D, from the completion of _TPM_INIT.
 - b. For TPMs that do comply with FIPS 140-2 SelfTest requirements, the TPM SHALL return the register contents within 50ms from the completion of _TPM_INIT.
 - c. For TPMs that comply with FIPS 140-3 SelfTest requirements, the TPM SHALL return the register contents within 200ms of _TPM_INIT.
 - d. For NV maintenance and recovery, the TPM SHALL return the register contents within 1 second of _TPM_INIT.

8.1.9 Locality support

Start of informative comment

With TPM 1.2 the so-called Locality concept was introduced as hardware-based authorization. A TPM 2.0 I2C Interface supports Locality because TPM 2.0 continues to use a TPM I2C Interface. To allow simplified implementations it is possible to implement an I2C-TPM with only one locality.

The indication of which localities are supported is done via the Interface Capability register (see section 8.3.5.10 of this document).

End of informative comment

1. An I2C-TPM compliant to this specification SHALL support one of the following options:
 - a. one locality (locality 0) or
 - b. 5 localities (locality 0 – locality 4).

8.1.10 GUARD_TIME

Start of informative comment

GUARD_TIME is the minimum elapsed time at the I2C master measured from the I2C STOP condition until the next I2C START condition. GUARD_TIME might be required by some I2C TPM implementations to recover between two separate access cycles.

The Interface Capability register (see section 8.3.5.10 of this document) indicates whether an I2C-TPM needs the GUARD_TIME and for which condition (write after read, write after write, read after write or read after read).

The default value for GUARD_TIME for all four conditions is 250 μ s. This value must be assumed until the actual value indicated by the I2C-TPM has been read from the corresponding fields in the Interface Capability register. Additionally, before the actual value for GUARD_TIME_Sr has been read from the Interface Capability register, the bus master must use the default value of 250 μ s for GUARD_TIME_Sr.

End of informative comment

8.2 Communication Protocol Fundamentals

Start of informative comment

Data transmission between the I2C-TPM and the HOST (as illustrated in Section 8.2.1 Layer Model) is done through the I2C interface of the I2C-TPM. Additionally, the I2C-TPM offers an indication that response data is available through a dedicated pin (PIRQ#). This low-active signal can also be used as an interrupt signal for other events as defined in section 8.3.5.5 TPM_INT_CAPABILITY of this document.

The communication flow between the HOST and the I2C-TPM is a strong dialog. That means the HOST needs to wait after a request for the corresponding response from the I2C-TPM before sending a new request.

End of informative comment

8.2.1 Layer Model

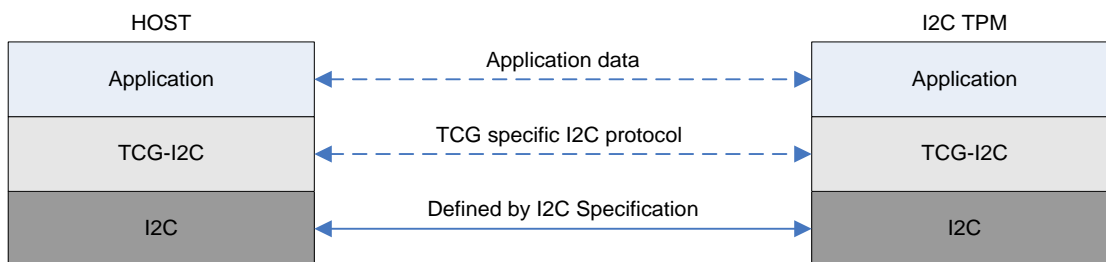


Figure 9 — Layer Model

8.2.2 Physical Layer I2C

Start of informative comment

The standardized physical layer is entirely defined in the I2C specification. Only a subset of those definitions is used for this protocol. See Section 8.1 of this document for details.

End of informative comment

8.2.2.1 I2C Protocol Usage Scenarios

Start of informative comment

Table 52 describes the behavior of I2C registers by locality. In PTP 1.06 the behavior of a TPM_DATA_CSUM register was modified for non-active localities. Previous versions of PTP defined this register as globally readable. PTP 1.06 restricts reading this register to the active Locality.

End of informative comment

Table 55 — Register Behavior Based on Locality Setting for I2C

TPM_ACCESS.activeLocality					
Set for This Locality		Set for another Locality		Not Set	
READ	WRITE	READ	WRITE	READ	WRITE
TPM_STS Registers					
TPM returns correct value	Fields updated	TPM returns 0xFF	TPM ignores the write	TPM returns 0xFF	TPM ignores the write
TPM_ACCESS Registers					
TPM returns correct value	Fields updated	TPM returns correct value	Fields updated	TPM returns correct value	Fields updated
TPM_DATA_FIFO Registers					
TPM returns correct data	TPM accepts data and command	TPM returns 0xFF	TPM ignores the write	TPM returns 0xFF	TPM ignores the write
TPM_HASH_START Register					
TPM returns 0xFF	TPM accepts command	TPM returns 0xFF	TPM ignores the write	TPM returns 0xFF	TPM accepts command and sets TPM_ACCESS.activeLocality for Locality 4
TPM_HASH_DATA Register					
TPM returns 0xFF	TPM accepts data	TPM returns 0xFF	TPM ignores the write	TPM returns 0xFF	TPM ignores the write
TPM_HASH_END Register					
TPM returns 0xFF	TPM accepts command and clears TPM_ACCESS.activeLocality for Locality 4	TPM returns 0xFF	TPM ignores the write	TPM returns 0xFF	TPM ignores the write
TPM_LOC_SEL Register					
TPM returns correct value	Fields updated	TPM returns correct value	Fields updated	TPM returns correct value	Fields updated
TPM_INT_ENABLE Register					
TPM returns correct value	Field updated	TPM returns correct value or 0xFF	Field updated	TPM returns correct value or 0xFF	Field updated
TPM_INT_STATUS Register					
TPM returns correct value	Interrupt cleared	TPM returns correct value	Interrupt cleared	TPM returns correct value	Interrupt cleared
TPM_INT_CAPABILITY Register					
TPM returns correct value	Read-only register	TPM returns correct value	Read-only register	TPM returns correct value	Read-only register
TPM_I2C_INTERFACE_CAPABILITY Register					
TPM returns correct value	Read-only register	TPM returns correct value	Read-only register	TPM returns correct value	Read-only register
TPM_I2C_DEVICE_ADDRESS Register					
TPM returns correct value or 0xFF	Fields updated	TPM returns correct value or 0xFF	Fields updated	TPM returns correct value or 0xFF	Fields updated

TPM_ACCESS.activeLocality					
Set for This Locality		Set for another Locality		Not Set	
READ	WRITE	READ	WRITE	READ	WRITE
TPM_CSUM_ENABLE Register					
TPM returns correct value	Fields updated	TPM returns correct value	Fields updated	TPM returns correct value	Fields updated
TPM_DATA_CSUM Register					
TPM returns correct value	Read-only register	TPM returns 0xFF	Read-only register	TPM returns 0xFF	Read-only register
TPM_DID_VID Register					
TPM returns correct value	Read-only register	TPM returns correct value	Read-only register	TPM returns correct value	Read-only register
TPM_RID Register					
TPM returns correct value	Read-only register	TPM returns correct value	Read-only register	TPM returns correct value	Read-only register

DRAFT

8.2.2.1.1 Regular Register Write

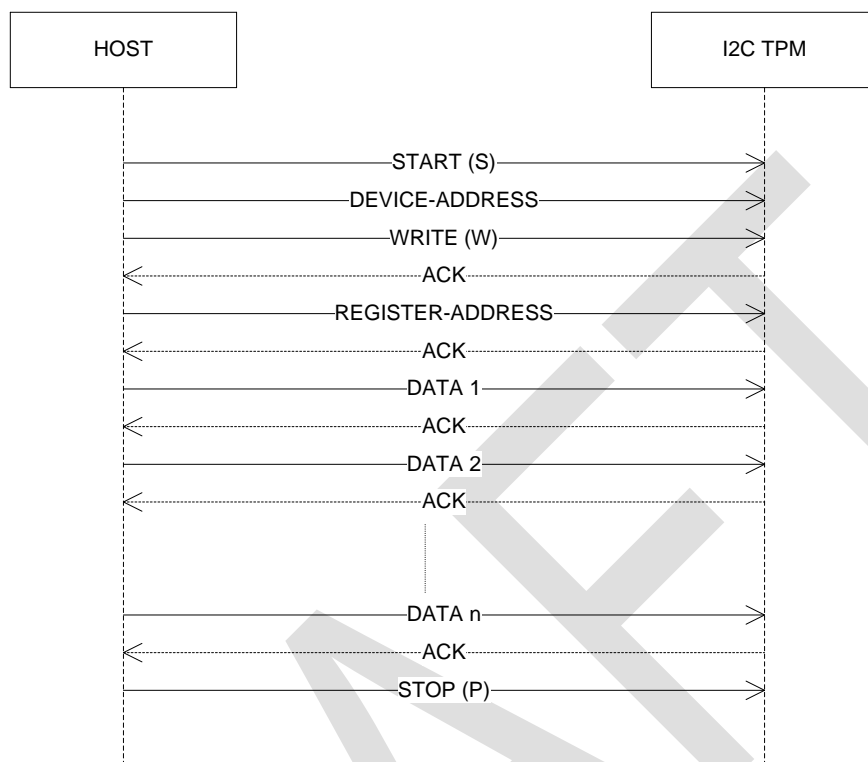


Figure 10 — Register write sequence on the I2C layer

8.2.2.1.2 Register Access with NACK

Start of informative comment

An address NACK is returned by the I2C-TPM when either an invalid I2C device address is used by the bus master or the I2C-TPM is currently unable to respond to the current bus cycle because of internal reasons. It is good practice to repeat the current cycle using the correct I2C device address. Note, the driver is not expected to understand the difference between a read access with an address NACK and a read access with a data NACK. The driver should be designed so that it attempts the access again with the correct TPM address following a wait time on the order of 250us, with a maximum retry count of 50. The address NACK for a register write is identical to the address NACK for a register read.

End of informative comment

1. If an I2C-TPM must return a NACK during the reception of data, it SHALL discard the data already written during this cycle.
2. A TPM SHALL return a NACK as a response to any I2C request to which it is unable to respond.

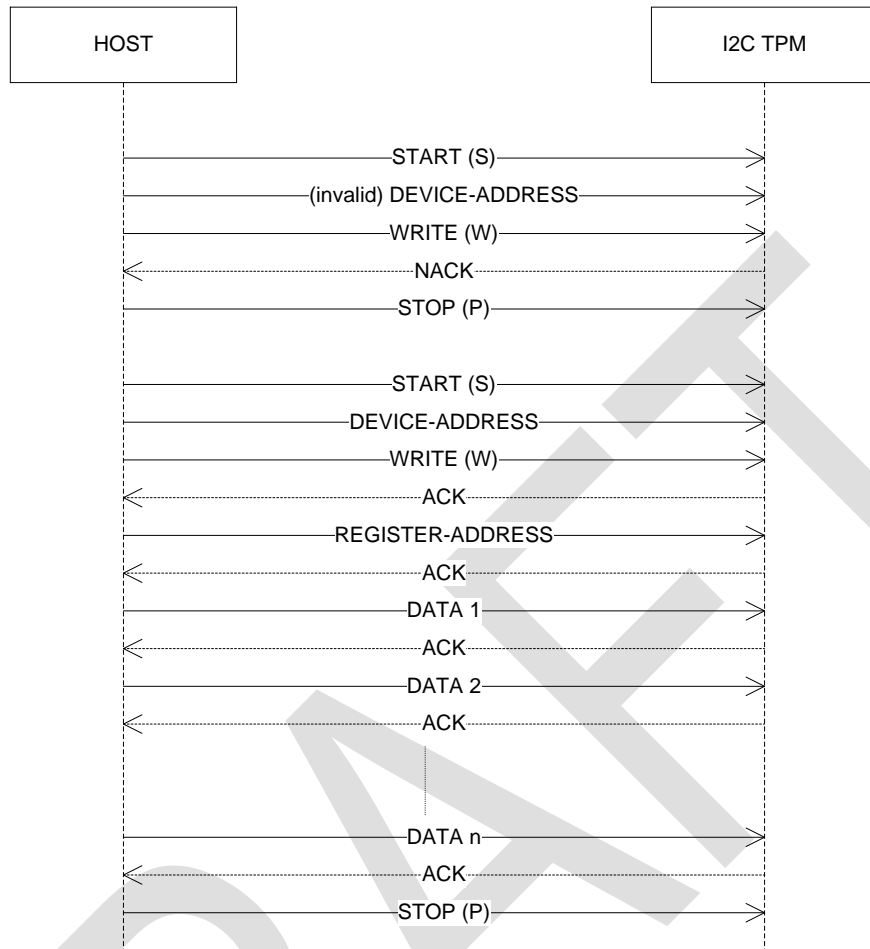


Figure 11 — Example Address NACK in a register write sequence on the I2C layer

8.2.2.1.3 Regular Register read

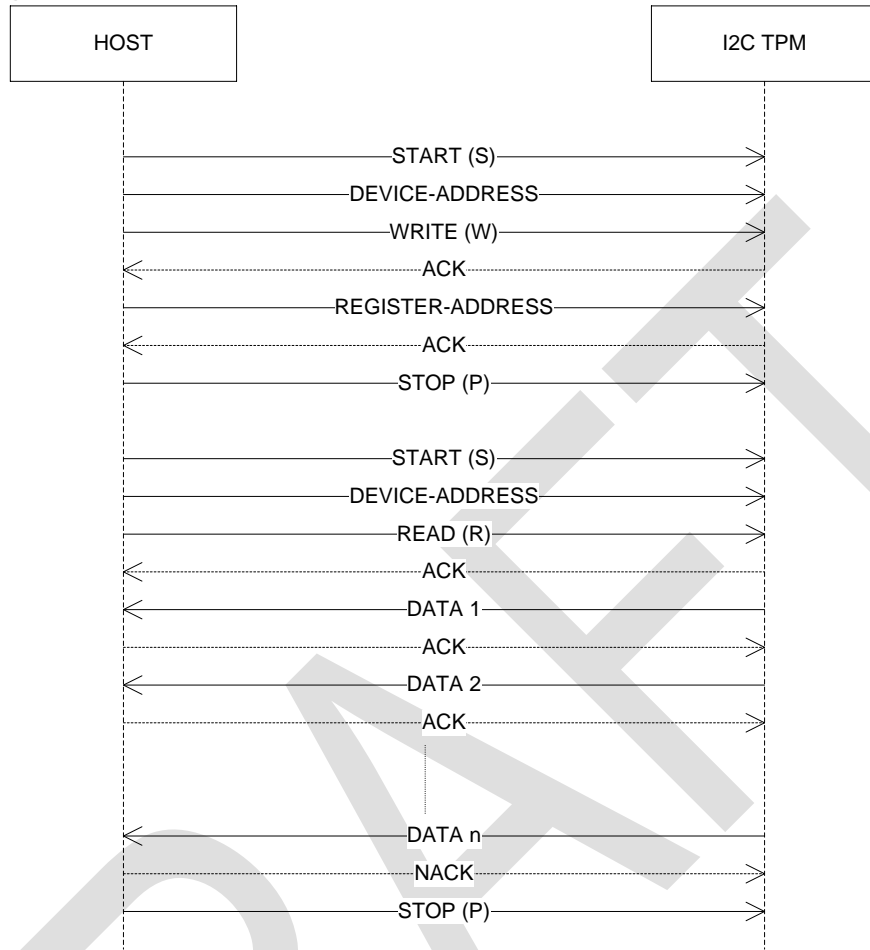


Figure 12 — Register read sequence on the I2C layer

8.2.2.1.4 Register read with repeated START

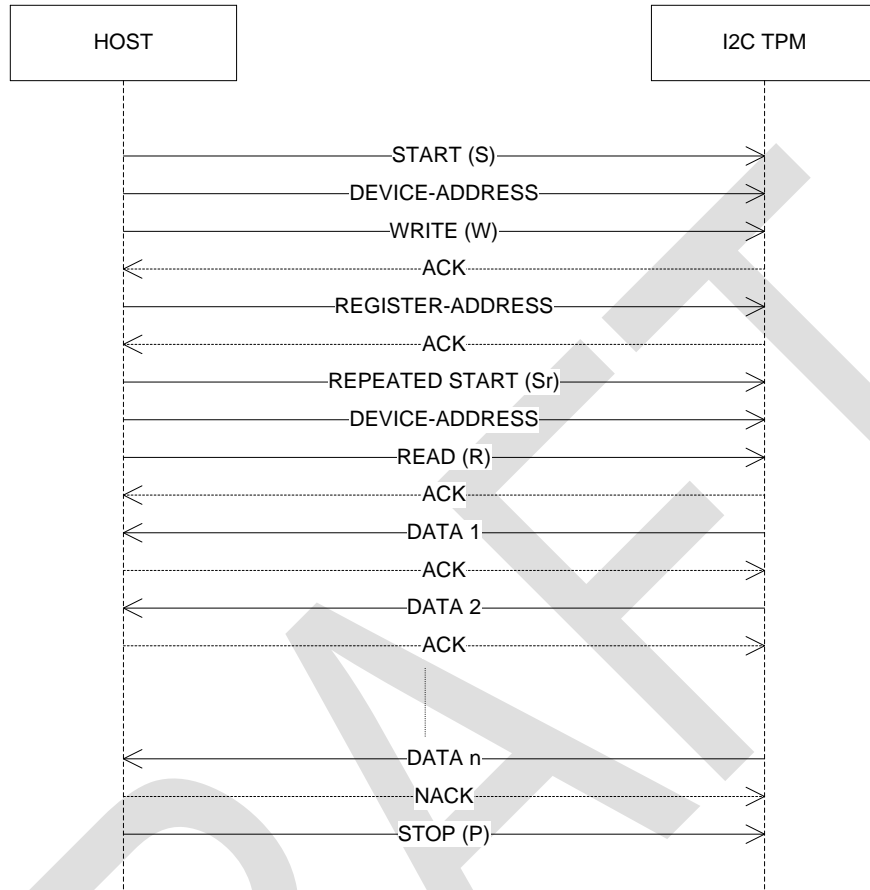


Figure 13 — Register read sequence on the I2C layer using repeated START (Sr)

8.2.2.1.5 Register read with **GUARD_TIME**

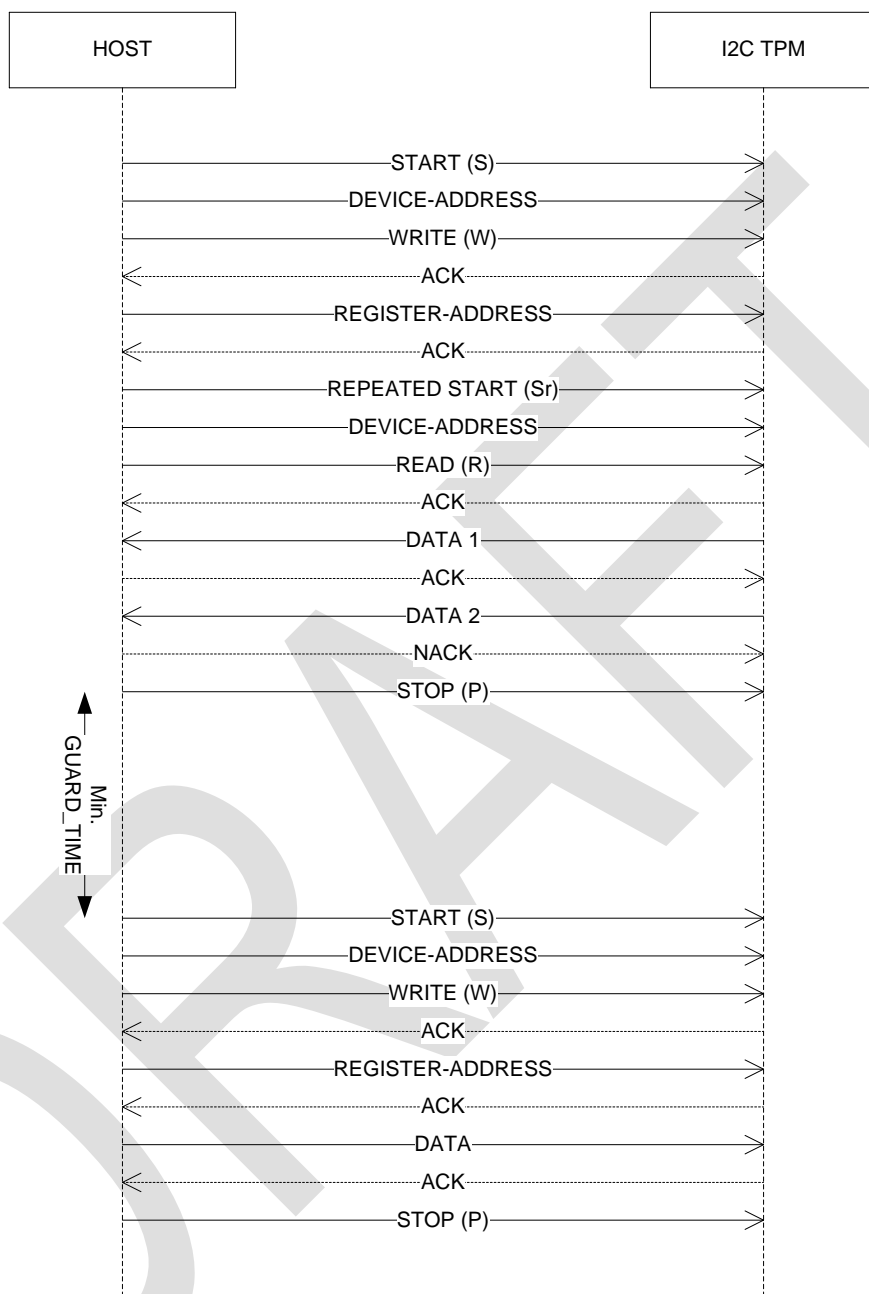


Figure 14 — Register read sequence with **GUARD_TIME** write after read on the I2C layer

8.3 Physical Layer TCG-I2C

Start of informative comment

The physical layer TCG-I2C is used to establish several sub-addresses below a single I2C device address as defined by the I2C specification [8]. These sub-addresses are defined as shown in the following sections of this document. The I2C slave uses different address locations for status, control, and data communication registers.

End of informative comment**8.3.1 Byte Ordering****Start of informative comment**

Multi-byte numeric values are stored and sent via the bus in little-endian order; for example, the first byte of a two-byte register is the LSB of the stored value while the second byte is its MSB. For example, if the master wants to read from a 4-byte register at 0x08 containing data0 at 0x08, data1 at 0x09, data2 at 0x0A and data3 at 0x0B the byte order on the bus looks as follows:

Register-Address:						0x08	0x09	0x0A	0x0B			
I2C:	S	DEVICE_ ADDRESS	W	REGISTER_ ADDRESS	Sr	DEVICE_ ADDRESS	R	Data0	Data1	Data2	Data3	P

End of informative comment**8.3.2 Overruns**

1. The I2C-TPM SHALL return a value of 0xFF in the following cases
 - a. On a read from an invalid register
 - b. On a read beyond the end of a register
2. If the master writes beyond the end of a register:
 - a. The I2C-TPM SHALL update the register designated by the start address.
 - b. The I2C-TPM MAY update additional, adjacent registers.

8.3.3 Handling of Multi-Byte Registers**Start of informative comment**

The I2C-TPM has various multi-byte registers (e.g., TPM_INT_ENABLE). Such registers are defined by a base address and a length. In all cases, except for a TPM_STS register, the access to such multi-byte registers is only possible from the register base address. For example, a write to a TPM_INT_ENABLE starts from base address 0x08 and consists of 4 bytes. A read from the same register also starts at the base address but may consist of 1 to 4 bytes (e.g., a read from 0x08 with only 1 byte would return bits 0 to 7 while a read from 0x09 may return 0xFF).

There is one exception concerning TPM_STS, which is divided into 3 parts. The 1st part with one byte at 0x18 (bits 0 to 7) may be written as a single byte. The 2nd part with two bytes at 0x19 is read only but may be written without effect (the write is ignored by the I2C-TPM). The 3rd part with one byte at 0x1B (bits 24 to 31) may also be written as a single byte. Consequently, a write to 0x18 may consist of 1 to 4 bytes. The same applies for a read, e.g., a read from 0x18 with 1 byte returns bits 0 to 7, a read from 0x18 with 4 bytes returns the entire TPM_STS register and a read from 0x1B with one byte returns bits 24 to 31.

End of informative comment

1. The I2C-TPM SHALL accept a write to a register base address with a data length equal to the length of the addressed register.
2. The I2C-TPM SHALL accept a read from a register base address and return the corresponding values using the rules in Table .
3. The I2C-TPM SHALL accept a single byte write to the addresses 0x18 or 0x1B (TPM_STS) and change the corresponding value.
4. The I2C-TPM SHALL accept a single byte read from the address 0x1B (TPM_STS) and return the corresponding value.
5. The I2C-TPM SHALL accept a 2-byte read from address 0x19 (TPM_STS) and return the corresponding values.
6. The behavior of the I2C-TPM for any other access is vendor specific.

8.3.4 I2C-TPM Localities

Start of informative comment

The I2C-TPM supports all localities as defined in Section 6.2 Locality, including the extended localities as defined in a TPM 2.0 Library Specification. Which locality currently accesses the I2C-TPM can be determined from the value in the locality selection register. Locality priority determines which locality takes precedence in cases where two or more localities request the I2C-TPM simultaneously. Table 56 shows the locality priorities with 1 as the highest priority and 6 as the lowest.

Note: The I2C definition is leveraged with other TCG platform workgroups utilizing TPM. PC Client TPMs do not use or define the extended localities referenced in Table 53 prior to PTP 1.06.

End of informative comment

Table 56 — TPM Locality Selection Register

Locality Priority	Locality Selection Register value	Locality	Value of locality modifier (see Section 6.2.1)	Mandatory (M) Optional (O)
5	0x00	0	0000 0001b	M
4	0x01	1	0000 0010b	O
3	0x02	2	0000 0100b	O
2	0x03	3	0000 1000b	O
1	0x04	4	0001 0000b	O

8.3.5 I2C-TPM Registers

Start of informative comment

The I2C-TPM registers are used to map the FIFO Interface for TPM 2.0 (see Section 6.5.2 FIFO Interface Requirements) to TPM 2.0 implementations using I2C as the Host interface. Those registers are established as sub-addresses below the single I2C device address as defined by the I2C specification [8].

The following registers are needed for the operation of a TPM at I2C with locality support. For a detailed description of registers (contents, and endianness of multi-byte registers) see Section 6.5.2.1 FIFO Register Space Addresses.

End of informative comment

Table 57 lists all registers of the I2C-TPM. A TPM_ACCESS register has multiple, separate, and unique instances, one per locality priority level (see Table 56). All other registers alias to a single register with the locality used to determine whether accesses are permitted. Note: the I2C register addresses may differ from the FIFO defined address space. This is due to the nature of I2C interface addressing.

Table 57 — I2C-TPM Register Overview

Address	Name	Length	Description	Master Access
TPM specific registers				
0x00	TPM_LOC_SEL	1	Selection of the locality of the current access	Read / Write
0x01 – 0x03	Reserved	N/A	Reads return 0xFF	N/A

Address	Name	Length	Description	Master Access
0x04	TPM_ACCESS	1	Used to gain ownership of a TPM for this locality	Read / Write
0x05 – 0x07	Reserved	N/A	Reads return 0xFF	N/A
0x08 – 0x0B	TPM_INT_ENABLE	4	Enables specific interrupts and has the global enable	Read / Write
0x0C – 0x0F	Reserved	N/A	Reads return 0xFF	N/A
0x10 – 0x13	TPM_INT_STATUS	4	Shows which interrupt has occurred	Read / Write
0x14 – 0x17	TPM_INT_CAPABILITY	4	Provides information about which interrupts this TPM supports	Read only
0x18 – 0x1B	TPM_STS	4	Contains general status details	Read / Write
0x1C – 0x1F	Reserved	N/A	Reads return 0xFF	N/A
0x20	TPM_HASH_END	1	This signals the end of the hash operation. Only available when locality 4 is selected	Write only
0x21 – 0x23	Reserved	N/A	Reads return 0xFF	N/A
0x24	TPM_DATA_FIFO	TPM_STS (burst-count)	Buffer to exchange the data for commands and responses with the HOST. For locality4 this is also aliased to TPM_HASH_DATA	Read / Write
0x25 – 0x27	Reserved	N/A	Reads return 0xFF	N/A
0x28	TPM_HASH_START	1	This signals the start of the hash operation. Only available when locality 4 is selected ²	Write only
0x29 – 0x2F	Reserved	N/A	Reads return 0xFF	N/A

² There are two ways to access TPM_HASH_START: the first is a write of 0x04 to TPM_LOC_SEL followed by a write to TPM_HASH_START; the second is a write of 0x04 to TPM_LOC_SEL followed by a write to TPM_ACCESS.requestUse, then a write to TPM_HASH_START.

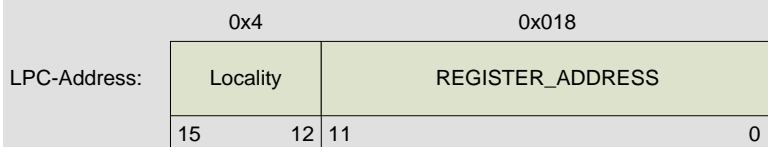
Address	Name	Length	Description	Master Access
0x30 – 0x33	TPM_I2C_INTERFACE_CAPABILITY	4	I2C Interface Capability Register	Read only
0x34 – 0x37	Reserved	N/A	Reads return 0xFF	N/A
0x38 – 0x39	TPM_I2C_DEVICE_ADDRESS	2	This register allows changing the I2C device address	Write only
0x3A – 0x3F	Reserved	N/A	Reads return 0xFF	N/A
0x40	TPM_DATA_CSUM_ENABLE	1	Enables the data checksum calculation and indication via a TPM_DATA_CSUM register	Read / Write
0x41 – 0x43	Reserved	N/A	Reads return 0xFF	N/A
0x44 – 0x45	TPM_DATA_CSUM	2	Contains the data checksum when enabled via DATA_CSUM_ENABLE	Read only
0x46 – 0x47	Reserved	N/A	Reads return 0xFF	N/A
0x48 – 0x4B	TPM_DID_VID	4	Vendor and device ID VID (bits 15:0) DID (bits 31:16)	Read only
0x4C	TPM_RID	1	Revision ID	Read only
0x4D – 0xFF	Reserved	N/A	Reads return 0xFF	N/A

8.3.5.1 TPM_LOC_SEL

Start of informative comment

This register is used to select the locality that accesses the I2C-TPM. While this register is just the indication of the locality that performs the current communication with the I2C TPM interface, it is still necessary to execute the process of getting access to a TPM via a TPM_ACCESS register to become the active locality. Or, in other words, the value in TPM_LOC_SEL may be considered to be the upper nibble of the LPC address.

For example, if locality4 wants to access the register address 0x18 the LPC address would have the value 0x4018:



The same access on the I2C TPM looks as follows:

I2C-Address:	0x04		0x18	
	TPM_LOC_SEL	REGISTER_ADDRESS		
	15	8	7	0

End of informative comment

Table 58 — TPM Locality Selection Register

Abbreviation:			TPM_LOC_SEL
General Description:			Indication of the accessing Locality
Bit Descriptions:			
7:0	Read/ Write	LocalitySelection	This register is sticky. Read returns the currently accessing locality. Write sets the new accessing locality. See Table 56 for allowed values, default 0x00

1. An I2C TPM SHALL maintain the value in this register until it is written with a new value.
2. An I2C TPM SHALL ignore all writes to this register during a D-RTM sequence, after receipt of HASH_START and prior to receipt of HASH_END.

8.3.5.2 TPM_ACCESS

Start of informative comment

A TPM_ACCESS register for I2C uses the same definition as that defined in Section 6.5.2.4 (Access Register).

End of informative comment

8.3.5.3 TPM_INT_ENABLE**Table 59 — Interrupt Enable**

Abbreviation:			TPM_INT_ENABLE
General Description:			Enables specific interrupts and has the global enable. An I2C TPM SHALL implement this register.
Bit Descriptions:			
31	Read/ Write	globalIntEnable	1 = Interrupts controlled by individual bits 0 = All interrupts disabled (default) Cleared to 0 on reset.
30:8		Reserved	Reads always return 0
7	Read/ Write	commandReadyEnable	1 = Enabled 0 = Disabled (default)
6:3		Reserved	Reads always return 0
2	Read/ Write	localityChangeIntEnable	1 = Enabled 0 = Disabled (default)
1	Read/ Write	stsValidIntEnable	1 = Enabled 0 = Disabled (default)
0	Read/ Write	dataAvailIntEnable	1 = Enabled 0 = Disabled (default)

8.3.5.4 TPM_INT_STATUS**Start of informative comment**

A TPM_INT_STATUS definition is the same as that defined for LPC. See Section 6.6.1.2.

End of informative comment**8.3.5.5 TPM_INT_CAPABILITY****Table 60 — Interrupt Capability**

Abbreviation:			TPM_INT_CAPABILITY
General Description:			Provides information about which interrupts an I2C TPM supports. An I2C TPM SHALL implement this register.
Bit Descriptions:			
31:8	Read Only	Reserved	Reads always return 0
7	Read Only	commandReadyIntSupport	Corresponds to TPM_INT_ENABLE.commandReadyEnable 1 = supported 0 = not supported
6:3	Read Only	Reserved	Reads always return 0
2	Read Only	LocalityChangeIntSupport	Corresponds to TPM_INT_ENABLE.localityChangeIntEnable. 1 = supported 0 = not supported
1	Read Only	stsValidIntSupport	Corresponds to TPM_INT_ENABLE.stsValidIntEnable 1 = supported 0 = not supported
0	Read Only	dataAvailIntSupport	Corresponds to TPM_INT_ENABLE.dataAvailIntEnable. This is a mandatory interrupt. 1 = supported 0 = not allowed

8.3.5.6 TPM_STS**Start of informative comment**

See Section 6.5.2.8 for a description of the state transition behavior.

Reading of the burstCount may be critical when read in single bytes because the low part might change when the high part is read (and vice versa). Therefore, it is strongly recommended to read the whole burstCount in one cycle.

End of informative comment**Table 61 — Status Register**

Abbreviation:			TPM_STS
General Description:			Contains general status details
Bit Descriptions:			
31:26	Read Only	Reserved	Reads always return 0.

Abbreviation:			TPM_STS
General Description:			Contains general status details
Bit Descriptions:			
25	Write Only	resetEstablishmentBit	Reads always return 0. Writes (0): Ignored. Writes (1): Reset TPM_ACCESS.tpmEstablished bit if the write occurs from Locality 3 or 4.
24	Write Only	commandCancel	Reads always return 0. A write of a 1 to this field after tpmGo and before dataAvail aborts the currently executing command, resulting in a response of TPM_RC_CANCELLED. A write of 1 to this field after dataAvail and before tpmGo is ignored by a TPM. Writes of 0 are ignored.
23:8	Read Only	burstCount	Indicates the number of bytes that an I2C TPM can return on reads or accept on writes without incurring wait states.
7	Read Only	stsValid	This field indicates that TPM_STS.dataAvail and TPM_STS.Expect contain a valid value.
6	Read/Write	commandReady	Read of 1 indicates a TPM is ready, write of 1 causes a TPM to transition its state.
5	Write Only	tpmGo	After Software has written a command to a TPM and sees that it was received correctly, Software SHALL write a 1 to this field to cause a TPM to execute that command.
4	Read Only	dataAvail	This field indicates that a TPM has data available as a response. When set to 1, Software MAY read the ReadFIFO. A TPM SHALL clear the field to 0 when it has returned all the data for the response. Valid indicator: TPM_STS.stsValid = 1
3	Read Only	Expect	An I2C TPM sets this field to a value of 1 when it expects another byte of data for a command. It clears this field to a value of 0 when it has received all the data it expects for that command, based on a TPM size field within the packet. Valid indicator: TPM_STS.stsValid = 1
2	Read Only	selfTestDone	This field indicates that a TPM has completed all self-test actions following a TPM2_SelfTest command. Read of 0 indicates self-test is not complete. Read of 1 indicates self-test is complete
1	Write Only	responseRetry	Software writes a 1 to this field to force a TPM to re-send the response. Reads always return 0.
0	Read Only	Reserved	Reads always return 0.

8.3.5.7 TPM_HASH_END

Start of informative comment

See Section 5.3 Locality Controlled Functions for a detailed description.

End of informative comment

8.3.5.8 TPM_DATA_FIFO

Start of informative comment

This register is the port used by a TPM to receive commands and return data and status to Software. TPM commands and return packets for commands are multiple bytes. Software should read a TPM_STS_x.burstCount field to determine how many consecutive bytes it can write or read without clock-stretching.

See Section 6.5.2.6 Data FIFO Register for a detailed description.

End of informative comment

8.3.5.9 TPM_HASH_START

Start of informative comment

See Section 5.3 Locality-Controlled Functions for a detailed description.

End of informative comment

8.3.5.10 TPM_I2C_INTERFACE_CAPABILITY

Table 62 — I2C Interface Capability Register

Abbreviation:			TPM_I2C_INTERFACE_CAPABILITY
General Description:			This register provides miscellaneous information about the interface capabilities of the I2C-TPM.
Bit Descriptions:			
31	Read Only	Reserved	Reads always return 0
30	Read Only	GUARD_TIME_Sr	Indicates whether the I2C-TPM needs a GUARD_TIME for repeated START conditions in addition to the conditions defined in Bits 20:17 of this register. 1 – GUARD_TIME needed between Last ACK/NACK to I2C repeated START 0 – No GUARD_TIME needed
29	Read Only	BurstCountStatic	Indicates whether a TPM_STS.burstCount field is dynamic or static 1 = TPM_STS.burstCount is static 0 = TPM_STS.burstCount is dynamic
I2C Device Address Change Capabilities			
28:27	Read Only	DevAdrChange	00 – Changing the I2C Device Address is not supported 01 – Changing the I2C Device Address is supported using a vendor defined mechanism 10 – Reserved (not allowed) 11 – Changing the I2C Device Address is supported using the TCG defined mechanism (see Section 8.3.5.15)
Locality support Capabilities			
26:25	Read Only	CapLocality	00 – This I2C TPM supports Locality 0 only. 01 – This I2C TPM supports 5 localities (0 – 4). 10 – This I2C TPM supports all localities (0 – 255). 11 – Reserved (not allowed)

Abbreviation:			TPM_I2C_INTERFACE_CAPABILITY
General Description:			This register provides miscellaneous information about the interface capabilities of the I2C-TPM.
Bit Descriptions:			
I2C Bus Speed Capabilities			
24	Read Only	HsModeSupport	1 – Support for I2C High-Speed Mode (Hs-mode) 0 – I2C High-Speed Mode not supported
23	Read Only	FmPlusSupport	1 – Support for I2C Fast Mode Plus (Fm+) 0 – I2C Fast Mode Plus not supported
22	Read Only	FmSupport	1 – Support for I2C Fast Mode (Fm, mandatory) 0 – Not allowed
21	Read Only	SmSupport	1 – Support for I2C Standard Mode (Sm, mandatory) 0 – Not allowed
<p>GUARD_TIME Capabilities: if any of the following bits are set to 1, the I2C-TPM requires a GUARD_TIME for the given condition Note: Please refer also to bit 30 in this register</p>			
20	Read Only	Read_Read	1 – GUARD_TIME needed between 2 subsequent I2C read operations (I2C STOP to I2C START) 0 – No GUARD_TIME needed
19	Read Only	Read_Write	1 – GUARD_TIME needed between a I2C read operation and the following I2C write operation (I2C STOP to I2C START) 0 – No GUARD_TIME needed
18	Read Only	Write_Read	1 – GUARD_TIME needed between a I2C write operation and the following I2C read operation (I2C STOP to I2C START) 0 – No GUARD_TIME needed
17	Read Only	Write_Write	1 – GUARD_TIME needed between 2 subsequent I2C write operations (I2C STOP to I2C START) 0 – No GUARD_TIME needed
16:9	Read Only	GUARD_TIME	The value in this register defines the GUARD_TIME needed by the I2C TPM if indicated in the bits 20:17 of this register. A value of 0 is only allowed if all bits 20:17 are set to 0. All other values represent the GUARD_TIME in μ s (e.g., 0x01 means 1 μ s and 0xFA means 250 μ s).

Abbreviation:		TPM_I2C_INTERFACE_CAPABILITY	
General Description:		This register provides miscellaneous information about the interface capabilities of the I2C-TPM.	
Bit Descriptions:			
Interface version detection			
8:7	Read Only	tpmFamily	TPM Family Identifier 00: TPM 1.2 Family 01: TPM 2.0 Family 10 – 11: Reserved
6:4	Read Only	InterfaceVersion	000: TCG I2C interface 1.0 as defined in this specification 001 – 111: Reserved
3:0	Read Only	InterfaceType	0010 – FIFO interface on I2C 0000 – Reserved – See section 6.4.2 0001 – Reserved – See section 6.4.2 1111 – Reserved – See section 6.4.2

8.3.5.11 TPM_DATA_CSUM_ENABLE

Start of informative comment

This register may be volatile and may not be preserved across a TPM_RESET or a TPM_RESTART. Device drivers may need to check this field and re-enable it.

End of informative comment

Table 63 — Data Checksum Enable Register

Abbreviation:		TPM_DATA_CSUM_ENABLE	
General Description:		Enables the data checksum calculation and indication via a TPM_DATA_CSUM register.	
Bit Descriptions:			
7:1		Reserved	Reads always return 0
0	Read/Write	dataCSumEnable	1 = Data Checksum enabled 0 = Data Checksum disabled (default)

1. An I2C TPM SHALL accept a write to this register after it has performed all power-on initialization (see Section 8.1.8 for details).
2. An I2C TPM SHALL accept a write to this register when it is in Idle or Ready state (see Section 6.5.2.5.1 for details).
3. An I2C TPM MAY NOT accept a write to this register when it is in Reception, Execution or Completion state.

8.3.5.12 TPM_DATA_CSUM

Table 64 — Data Checksum Register

Abbreviation:		TPM_DATA_CSUM	
General Description:		Contains the data checksum when enabled	
Bit Descriptions:			
15:0	Read only	DataChecksum	Read returns the Checksum of the entire command data at the end of the command transmission or the Checksum of the entire response data at the end of the response transmission. Default: 0x00

1. An I2C TPM SHALL use CRC-CCITT (KERMIT) for the calculation of the data checksum (see Section 9 for further details). The Kermit parameters are as follows:
 - a. The generator polynomial is $0x1021 (x^{16} + x^{12} + x^5 + 1)$
 - b. The initialization value is 0x0000.
 - c. Reflection of input data: TRUE
 - d. Reflection of output data: TRUE
 - e. Final XOR: 0x0000
2. Test vectors:
 - a. The CRC value for the ASCII string "123456789" is 0x8921.
A 2-byte read from 0x44 will return
Data0 (LSB) = 0x21 and Data1 (MSB) = 0x89.
 - b. The CRC value for the ASCII string "1122334455" is 0xD367.
A 2-byte read from 0x44 will return
Data0 (LSB) = 0x67 and Data1 (MSB) = 0xD3.
 - c. The CRC value for the HEX string 00 C1 00 00 00 0C 00 00 00 99 00 01₁₆ (TPM2_StartUp(ST_CLEAR)) is 0xFBBF.
A 2-byte read from 0x44 will return
Data0 (LSB) = 0xBF and Data1 (MSB) = 0xFB.
 - d. The CRC value for the HEX string 80 01 00 00 00 0C 00 00 01 44 00 00₁₆ (TPM2_StartUp(TPM_SU_CLEAR)) is 0x6733.
A 2-byte read from 0x44 will return
Data0 (LSB) = 0x33 and Data1 (MSB) = 0x67.
3. If enabled via a TPM_DATA_CSUM_ENABLE register:
 - a. An I2C TPM SHALL calculate a checksum over the entire command.
 - b. An I2C TPM SHALL calculate a checksum over the entire response.
 - c. An I2C TPM SHALL update the command checksum after reception of the last command byte and before the transition of TPM_STS.Expect from 1 to 0.
The I2CTPM SHALL maintain the command checksum from the transition of TPM_STS.Expect from 1 to 0 until TPM_STS.tpmGo is set to 1.
 - d. An I2C TPM SHALL update the response checksum after the last response byte has been read and before the transition of TPM_STS.dataAvail from 1 to 0.
An I2C TPM SHALL maintain the response checksum from the transition of TPM_STS.dataAvail from 1 to 0 until Host writes a 1 to TPM_STS.commandReady.

8.3.5.13 TPM_DID_VID

Start of informative comment

See section 6.4.1.1 DID/VID Register.

End of informative comment**8.3.5.14 TPM_RID****Start of informative comment**

See section 6.4.1.2 RID Register.

End of informative comment**8.3.5.15 TPM_I2C_DEVICE_ADDRESS****Start of informative comment**

For I2C devices connected to the same bus each device must have its own unique I2C device address to avoid a bus conflict. There are 3 different possibilities:

- A device has a fixed address which can't be changed
- A device address may be configured via dedicated pins
- A device address may be changed using a dedicated command or register

The I2C TPM specification defines a register mechanism to change the I2C device address in situations where the bus needs to be shared with other devices that can't be re-configured. If that is the case a simple write access to a TPM_I2C_DEVICE_ADDRESS register with the new deviceAddress and (this is recommended) the makePersistent bit set will solve such an address conflict.

There may be situations where the new deviceAddress is only needed for one power cycle. In such cases the makePersistent bit should be set to 0. If makePersistent is 1: a write to deviceAddress sets a new address that will be saved to NVM immediately and becomes active immediately. If makePersistent is 0; the new deviceAddress becomes immediately active but is not preserved across a reset/power cycle.

End of informative comment**Table 65 — I2C Device Address Register**

Abbreviation:			TPM_I2C_DEVICE_ADDRESS
General Description:			This register holds the I2C device address.
Bit Descriptions:			
15	Write only	makePersistent	1 = Persistent device address defined by bits 6:0 0= Volatile device address defined by bits 6:0, lost after reset.
14:7		Reserved	Reads always return 0
6:0	Write only	deviceAddress	I2C device address Reads return either the current I2C device address or FFh. Writes set the new I2C device address effective with the next I2C master access. Default: 0x2E

8.3.6 Interface Locality Usage per Register**Start of informative comment**

Table 55 shows how an I2C TPM responds to accesses to each of the interface registers based on locality settings for the FIFO interface.

End of informative comment

8.3.7 TCG-I2C Protocol Usage Scenarios

8.3.7.1 Simple access to TPM_ACCESS

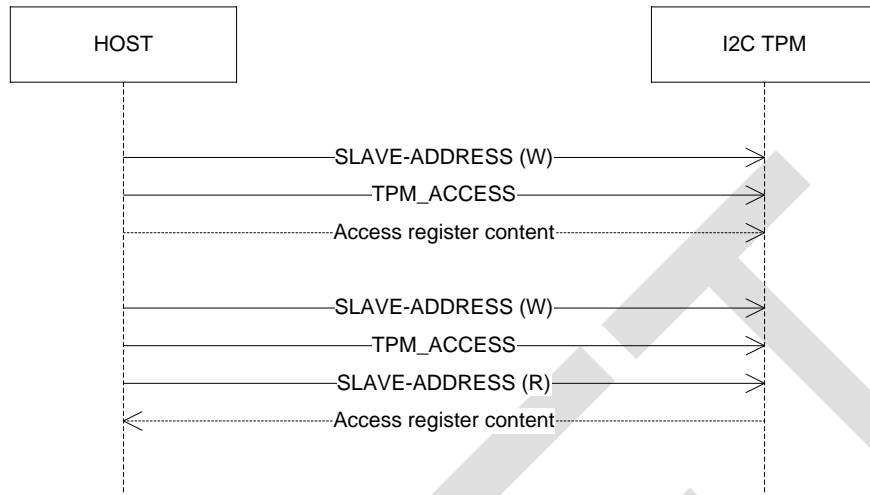


Figure 15 — Write / Read TPM_ACCESS register without locality selection

8.3.7.2 Access to TPM_ACCESS from Locality 0 only

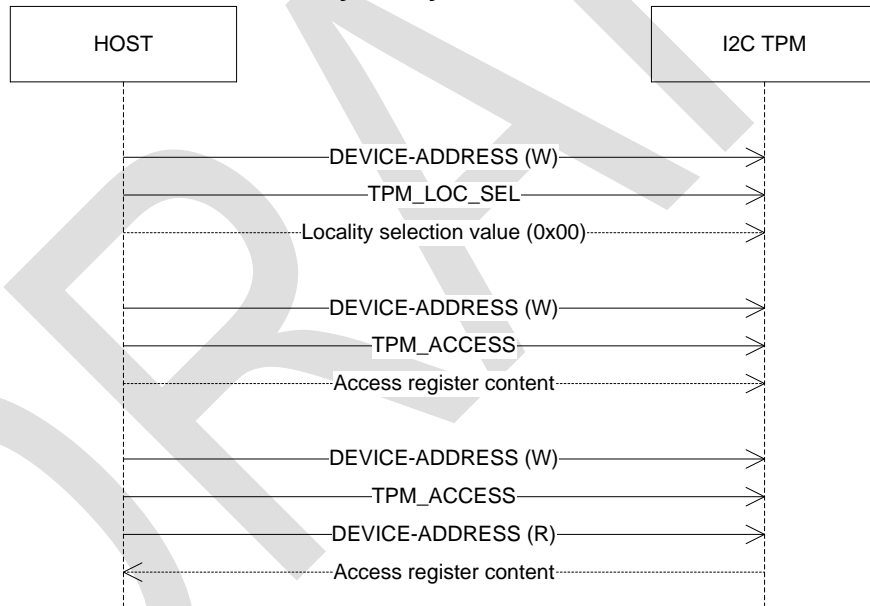


Figure 16 — Write / Read TPM_ACCESS register from Locality 0

8.3.7.3 Access to TPM_ACCESS from Locality 0 and 2

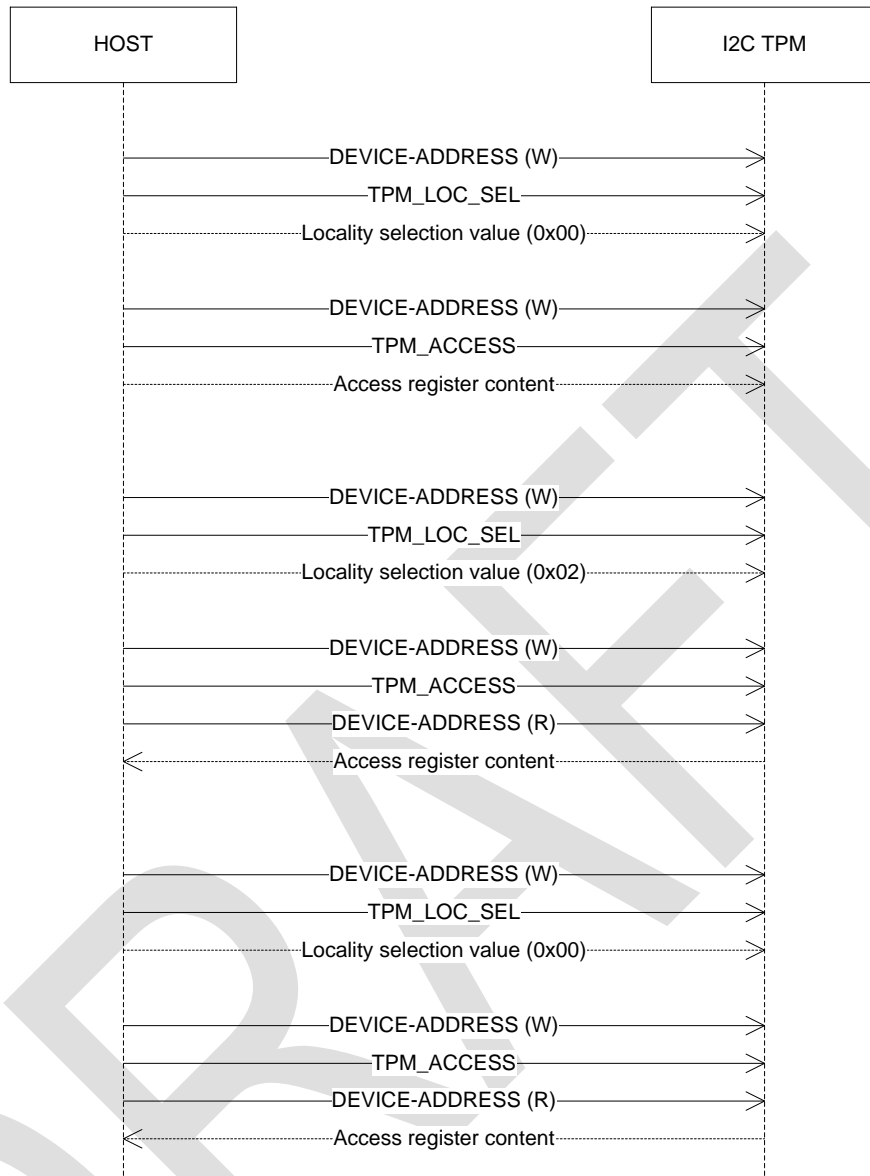


Figure 17 — Write / Read TPM_ACCESS register from Locality 0 and 2

8.3.7.4 Access to TPM_STS from Locality 0

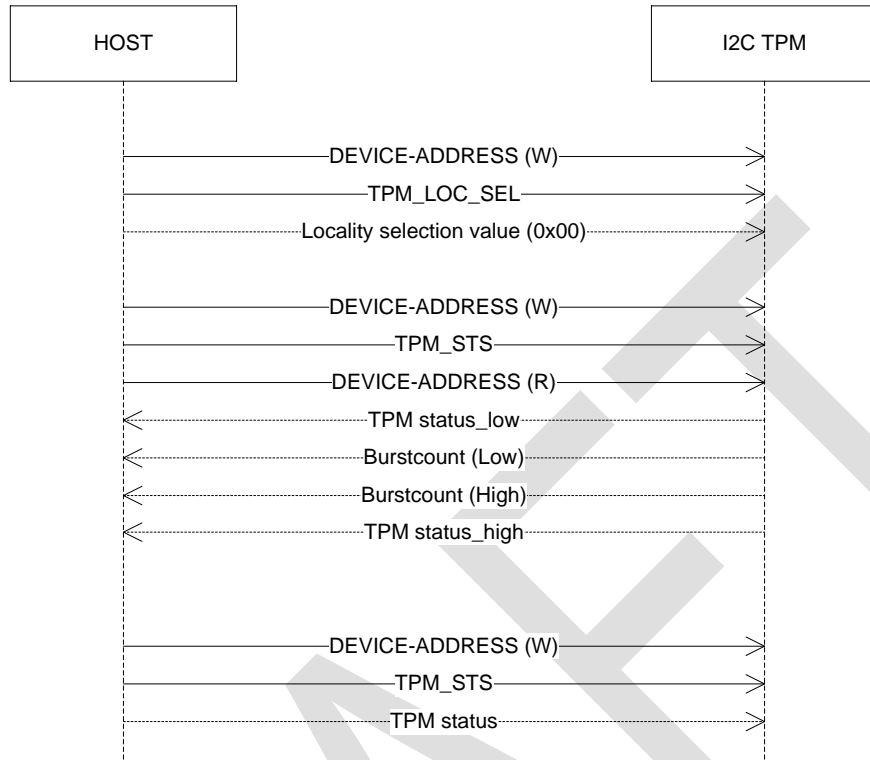


Figure 18 — Read / Write TPM_STS register(s) from Locality 0

8.3.7.5 Read from TPM_DATA_FIFO from Locality 0

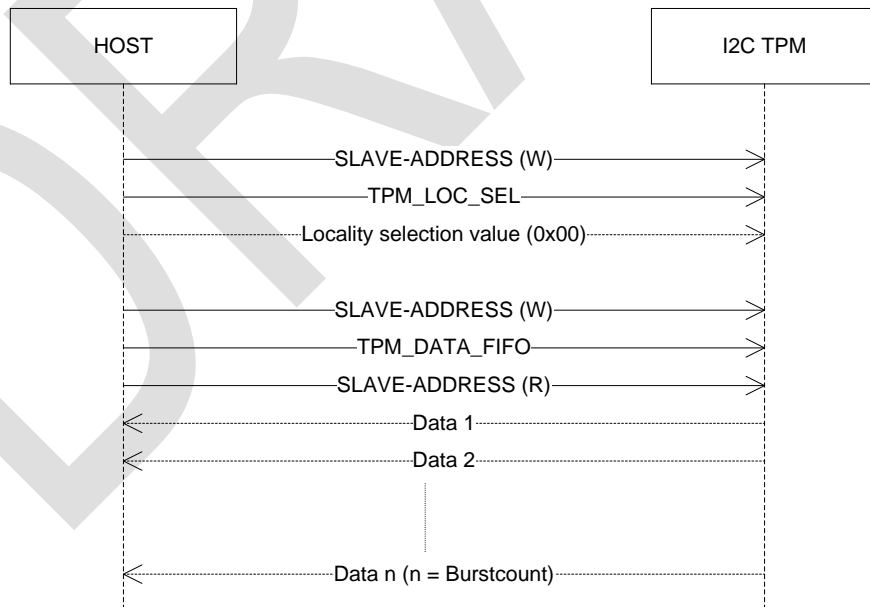


Figure 19 — Read TPM_DATA_FIFO

8.3.7.6 Write to TPM_DATA_FIFO from Locality 0

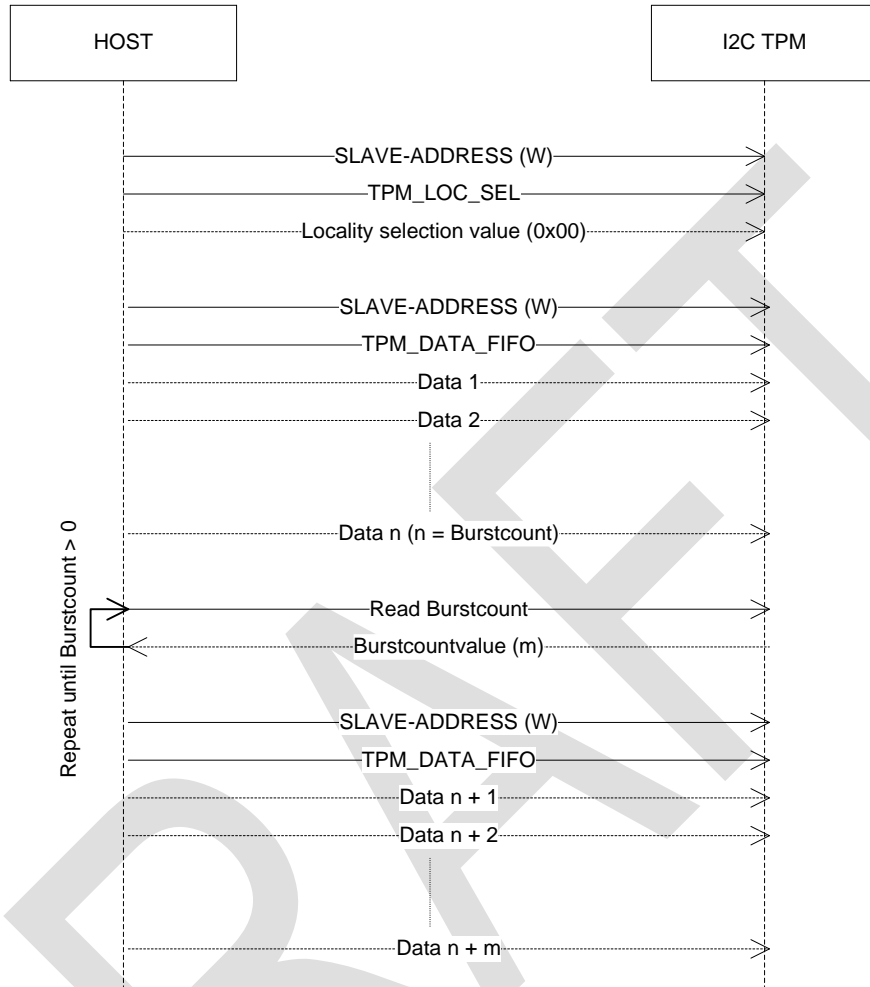


Figure 20 — Write TPM_DATA_FIFO

9 TPM Hardware Implementation

Start of informative comment

Hardware implementations of a TPM as a device in a PC Client platform require the careful consideration of some key elements. This section provides guidance for a TPM vendor's hardware implementation of a TPM and for the motherboard manufacturers designing a TPM into a PC Client platform. Section 9.1 TPM Packaging is targeted at TPM vendors, providing for a standardized package and pin-out that allows for form and fit compatibility across multiple TPM vendors, providing the greatest design flexibility for both TPM vendors and motherboard manufacturers. Section 10 Hardware Implementation of a TPM in a PC Client Platform is targeted at motherboard manufacturers, providing a collection of the critical hardware elements necessary to implement a TPM in a PC Client system.

End of informative comment

9.1 TPM Packaging

Start of informative comment

A standard package provides TPM and motherboard manufacturers the convenience and cost savings of not having to define from scratch the packaging and pin-out for a TPM. This packaging and pin-out recommendation is provided as a convenience for either an end product, or as a basis for extension or modification. It is recognized that individual environments may dictate other schemes; therefore, implementation of this section is optional, and any deviance will not detract from a platform's claim to adherence to this specification.

End of informative comment

1. To claim compliance to this section of this specification, a TPM SHALL use both the packaging and pin out as defined in this section:
 - a. It SHALL be said to use the "Packaging as specified in a TPM Packaging Section of the TCG PC Client Specific Platform TPM Profile for TPM 2.0 (PTP)".
 - b. It SHALL be designed using one of the following packages:
 - i. A 28-pin TSSOP using 9.6 mm plastic length (with 0.65 mm lead pitch) by 6.1 mm or 4.4 mm plastic width.
 - ii. A 32-pin QFN using 5mm width x 5mm length.
2. If a TPM does not use either the packaging or pin out specified in this section:
 - a. It SHALL NOT claim compliance to this section of this specification.
 - b. A TPM manufacturer SHALL provide documentation to the platform manufacturer regarding the package and pin out, including the GPIO-Express-00 pin's electrical characteristics.

GPIO/SM_DAT/I2C_SDA	1	28	LPCPD#
GPIO/SM_CLK/I2C_SCL	2	27	SIRQ
VNC	3	26	LAD0/MISO
GND	4	25	GND
VS	5	24	VDD
GPIO-Express-00	6	23	LAD1/MOSI
PP/GPIO	7	22	LFRAME#/SPI_CS#
TestI	8	21	LCLK/SPI_CLK
TestBI/BADD/GPIO	9	20	LAD2/SPI_PIRQ#/I2C_PIRQ#
VDD	10	19	VDD
GND	11	18	GND
VBAT	12	17	LAD3
xtalI/32k in	13	16	LRESET#/SPI_RST#
xtalO	14	15	CLKRUN#/GPIO/I2C_PIRQ#

Figure 21 — TPM Combo TSSOP-28 Pin Out

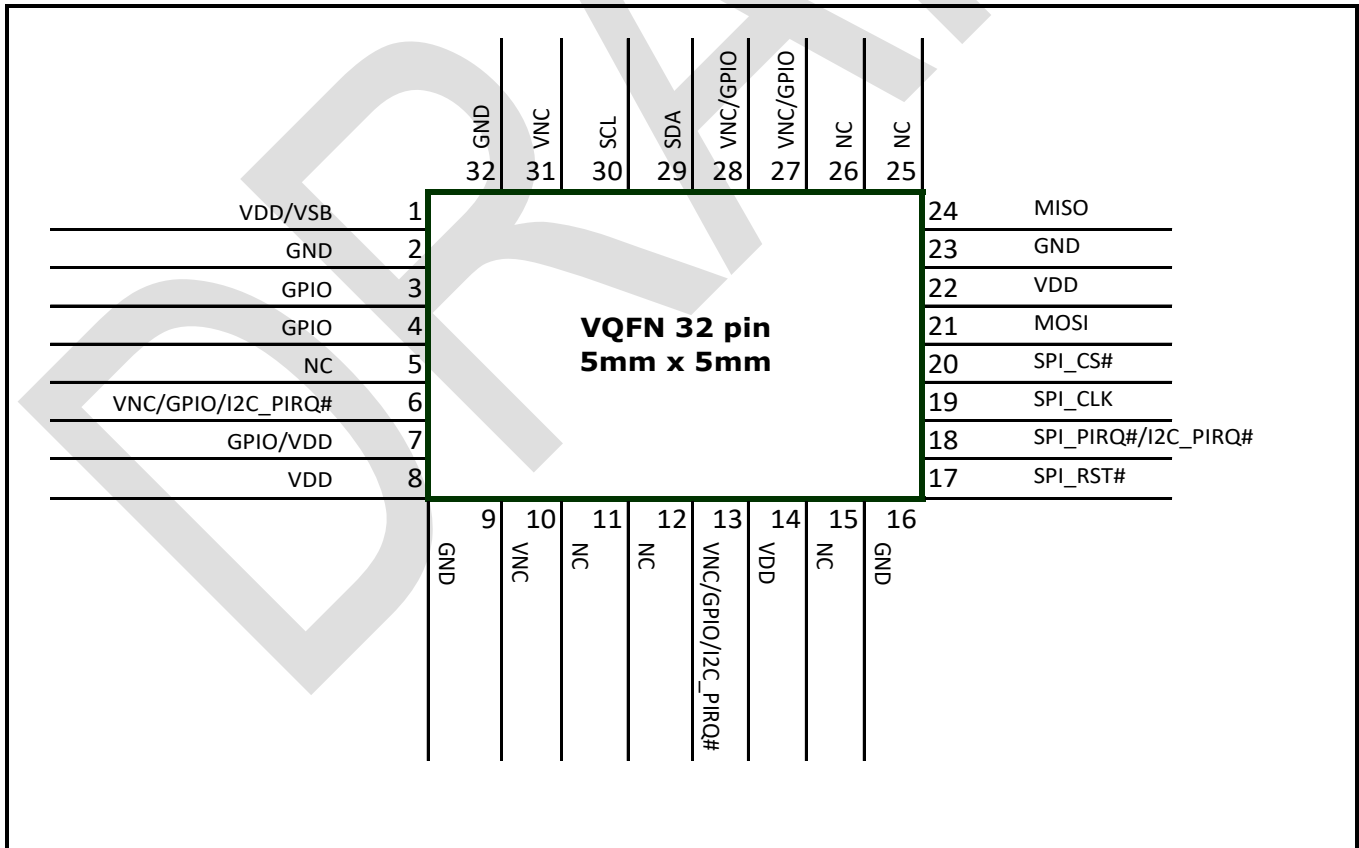


Figure 22 —TPM SPI QFN-32 Pin Out

- Using the pin-out defined in Figure 21 TPM Combo TSSOP 28 Pin Out, the pins SHALL be assigned as defined in Table 66. Using the pin-out defined in Figure 22, the pins SHALL be assigned as defined in Table 68.
- For SPI and I2C, TPM Packaging SHALL be implemented with a PIRQ# pin in one of the pins indicated in Tables 58, 59 and 60. It SHALL NOT be implemented in more than one pin.

Start of informative comment

The pins which are marked “M” for “mandatory” in Table 66 and Table 68 are required to be implemented. Those which are marked “O” for “optional” are not required for claims of adherence, but if implemented, must be implemented per Table 66 and Table 68.

End of informative comment**Table 66 — TSSOP-28 Pin Assignments**

Signal	Pin(s)	Type	Description	LPC pin-assignments	SPI pin-assignments
LAD3	17	BI	As defined in the LPC Interface Specification	M	O
LAD2/SPI_PIRQ#	20	BI/O	LAD 2: As defined in the LPC Interface Specification SPI_PIRQ#: SPI Interrupt, active low, open collector	LAD2 – M	PIRQ#-M
LAD1/MOSI	23	BI	LAD1: As defined in the LPC Interface Specification MOSI: As defined in Section 7.4 SPI Hardware Protocol	M	MOSI-M
LAD0/MISO	26	BI	LADO: As defined in the LPC Interface Specification	M	MISO-M
LPCPD#	28	I	Implementation of this pin SHALL allow for the pin to be strapped HIGH.	O	O
LCLK/SPI_CLK	21	I	LCLK: As defined in the LPC Interface Specification. SPI_CLK: As defined in Section 7.4.1 Clocking	LCLK-M	SPI_CLK-M
LFRAME#/SPI_CS#	22	I	LFRAME#: As defined in the LPC Interface Specification SPI_CS#: As defined in Section 7.4 SPI Hardware Protocol	LFRAME-M	SPI_CS#-M
LRESET#/SPI_RST#	16	I	LRESET#: As defined in the LPC Interface Specification SPI_RST#: Active Low	LRESET#-M	SPI_RST#-M
SERIRQ	27	BI	As defined in the LPC Interface Specification	M	O

Signal	Pin(s)	Type	Description	LPC pin-assignments	SPI pin-assignments
CLKRUN#/GPIO	15	BI	Same as PCI CLKRUN#. Active Low, internal pull-down. Only needed by peripherals that need DMA or bus mastering in a system that can stop the PCI bus (generally mobile devices). Implementation of CLKRUN# is TPM and chipset vendor specific GPIO will default to low.	O	O
PP/GPIO	7	I,BI	Physical Presence, active high, internal pull-down. Used to indicate Physical Presence to a TPM. GPIO will default to low	O	O
XTALI/32k in	13	I	32 kHz crystal input or 32 kHz clock input	O	O
XTALO	14	O	32 kHz crystal output	O	O
GPIO/SM_CLK	2	BI	Defaults as a GPIO. GPIO will default high. Also used as System Management Bus (SMB) Clock signal	O	O
GPIO/SM_DAT	1	BI	Defaults as a GPIO. GPIO will default high. Also used as System Management Bus (SMB) Data signal.	O	O
GPIO-Express-00	6	BI	GPIO assigned to TPM_NV_INDEX_GPIO_00, internal pull-up. Open-Collector output (when configured as output).	O	O
VNC	3		Vendor-controlled No Connect. This pin will be defined by a TPM vendor or can be a GPIO. There is no defined default state for this signal.	O	O
TESTI	8	I	This pin will be pulled low on the motherboard. Pull high to enable Test mode. Pull low to disable Test mode and enable GPIO/BADD on pin 9(TESTBI).	O	O
TESTBI/ BADD/GPIO	9	I/O	TESTBI: Test port. Internal pull-up If TESTI is pulled low, TESTBI acts as a GPIO and (optionally) BADD. GPIO will default high. BADD (optional, defaults high, use external pull-down to signal "low") can be used to select the legacy I/O base address. This logic is manufacturer specific, as well as the selected addresses. Setting is read at Startup.	O	O
Power					

Signal	Pin(s)	Type	Description	LPC pin-assignments	SPI pin-assignments
VDD	10, 19 24	I	This is either a 3.3 volt or 1.8V DC power rail supplied by the motherboard to the module. The maximum current from this interface is 250 mA. Available from S0-S2.	M	M
GND	4, 11, 18, 25	I	Zero volts. Expected to be connected to main motherboard ground.	M	M
VBAT	12	I	Battery input, may be 3.3V. Available from S0-S5 and in G3 state.	O	O
VSB	5	I	Standby DC power rail, may be 3.3V or 1.8V. Available from S0-S5.	O	O

Table 67 — TSSOP 28 I2C Pin Assignments

Signal	Pin(s)	Type	Description	I2C Pin assignment
GPIO/I2C_PIRQ#	20	BI/O	I2C_PIRQ#: Optional location for I2C PIRQ#, active low, open drain. Note: I2C interrupt must be present on either this pin or pin 15.	O ³
GPIO	6, 7, 9	I/O	GPIO will default high.	O
I2C_SCL	2	BI	I2C SCL signal; open drain	M
I2C_SDA	1	BI	I2C SDA signal; open drain	M
I2C_PIRQ#	15	BI	I2C_PIRQ#: Optional location for I2C PIRQ#, active low, open drain. Note: I2C interrupt must be present on either this pin or pin 20.	O ³
VNC	3, 8, 13, 14, 16, 17, 21, 22, 23, 26, 27, 28		Vendor-controlled No Connect. This pin will be defined by a TPM vendor or can be a GPIO. There is no defined default state for this signal.	O
Power				
VDD	10, 19, 24	I	This is either a 3.3 volt or 1.8V DC power rail supplied by the motherboard to the module. The maximum current from this interface is 250 mA. Available from S0-S2.	M
GND	4, 11,18, 25	I	Zero volts. Expected to be connected to main motherboard ground.	M
VBAT	12	I	Battery input, may be 3.3V. Available from S0-S5 and in G3 state.	O
VSB	5	I	Standby DC power rail, may be 3.3V or 1.8V. Available from S0-S5.	O

³ Depending on a TPM implementation, the I2C_PIRQ# pin will either be located on Pin 15 or Pin 20, not both. See TPM vendor's datasheet.

Table 68 — QFN-32 Pin Assignments

Signal	Pin(s)	Type	Description	Pin-assignments
SPI_PIRQ#/I2C_PIRQ#	18	BI/O	PIRQ#: SPI Interrupt, active low, open collector, I2C_PIRQ#: Optional location for I2C PIRQ#, active low, open drain. Note: I2C interrupt must be present on either this pin or pin 6.	M-SPI O-I2C ⁴
SPI_CLK	19	I	SPI_CLK: As defined in Section 7.4.1 Clocking	M
SPI_CS#	20	I	SPI_CS#: As defined in Section 7.4 SPI Hardware Protocol	M
SPI_RST#/LRESET#	17	I	SPI_RST#: Active Low LRESET#: Active Low	M
GPIO	3, 4	BI	GPIO defaults to input with weak termination. Level defined by vendor.	O
VNC/GPIO/I2C_PIRQ#	6	BI/O	I2C_PIRQ#: Optional location for I2C PIRQ#, active low, open drain. Note: I2C interrupt must be present on either this pin or pin 18.	O-I2C ⁴
VNC/GPIO/I2C_PIRQ#	13	BI/O	I2C_PIRQ#: Optional location for I2C PIRQ#, active low, open drain. Note: I2C interrupt must be present on either this pin or pin 18.	O-I2C ⁴
VNC/GPIO	27, 28	I,BI	Vendor defined no-connect. GPIO will default input with weak termination. Level defined by vendor.	O
MOSI	21	BI	MOSI – As defined in Section 7.4 SPI Hardware Protocol	M
MISO	24	BI	MISO – As defined in Section 7.4 SPI Hardware Protocol	M
SDA	29	I/O	I2C Data pin, as defined in Section 8.1.6 Pull-up Resistors. If implemented SCL must also be implemented.	O
SCL	30	I/O	I2C Clock pin, as defined in Section 8.1.6 Pull-up Resistors. If implemented SDA must also be implemented.	O
VNC	10, 31		Vendor-controlled No Connect. This pin will be defined by a TPM vendor or can be a GPIO. There is no defined default state for this signal.	O
Power				

⁴ Depending on a TPM implementation, the I2C_PIRQ# pin will either be located on Pin 6 or Pin 18, not both. See TPM vendor's datasheet.

Signal	Pin(s)	Type	Description	Pin-assignments
VDD	8,14, 22	I	This is either a 3.3-volt or 1.8-volt DC power rail supplied by the motherboard to the module. The maximum current from this interface is 250 mA. Available from S0-S2.	M
VDD/USB	1	I	This is either a 3.3-volt or 1.8-volt DC power rail supplied by the motherboard to the module. The maximum current from this interface is 250 mA. Available from S0-S2. If defined as VSB, this is a 3.3V supply.	M
GND	2, 9, 16, 23, 32	I	Zero volts. Expected to be connected to main motherboard ground.	M
NC	5, 11, 12, 15, 25, 26, 27, 28		No connection	O
GPIO/Power	7	I	GPIO defaults to input with weak termination, e.g., Physical Presence Power is vendor defined	O

10 Platform Specific Hardware Implementation of a TPM in a PC Client Platform

10.1 Electrical Connections for a Discrete TPM

Start of informative comment

A TPM in the PC Client platform serves as part of the Roots of Trust. As such, the hardware implementation of a TPM on the motherboard must account for how a TPM is connected to the other components of the platform that form the trust chain, such as the CPU. It is important that a TPM reset, clock and power signals support a TPM's function as the RTM and RTR and cannot be easily circumvented. Motherboard manufacturers should take care to ensure that the physical connections and routing minimize the possibility of attacking the S-CRTM and D-RTM.

End of informative comment

1. The `_TPM_INIT` (LRESET#/SPI_RST#) signal SHALL be connected to the platform CPU Reset signal such that a TPM complies with the behavior described in Section 2.2.7 HOST Platform and TPM Reset of the PC Client Platform Firmware Profile for TPM 2.0 Systems.
2. A TPM's main power pins (VDD) SHALL be connected such that a TPM is powered during ACPI states S0-S2 and MAY be powered in S3-S5.
3. If a TPM implements the optional VBAT and/or VSB pins, the pins MAY be connected to a battery or auxiliary power source. The motherboard manufacturer SHOULD consult their TPM documentation.
4. If an SPI TPM is implemented using the recommended packaging as defined in Figure 22 and Table 68, a TPM's SPI bus SHALL be connected so that:
 - a. A TPM has a dedicated chip select (SPI_CS#).
 - b. A TPM's SPI_RST# is connected directly to the platform's RST#, so that it cannot be controlled independently of assertion of the reset signal to the CPU or SOC.
5. An SPI TPM SHALL be implemented in a platform such that a TPM is only accessed when a TPM's SPI_CS# is asserted by the chipset.

NOTE: Locality 4 accesses to the hardware hash registers may be accessed via an implementation specific mechanism other than MMIO, but these accesses still obey this rule by virtue of being in a TPM's address range.

6. The Platform SHALL provide a hardware mechanism, e.g., a hardware-based strap, to configure the platform's TPM and chipset to support the SPI interface.

10.1.1 SPI Platform Design Notes

Start of informative comment

This section provides guidelines for platform OEMs and ISVs to aid in the design of platforms and Software using an SPI TPM. The following sections are informative only, as they describe recommended behavior.

End of informative comment

10.1.2 Software Interface to SPI-TPM

Start of informative comment

The SPI interface has been architected to be transparent to the driver and application layers in a TPM-enabled Software stack. Some SPI properties produce different results than LPC in cases where Software does not follow good design practice. In these cases, this specification addresses a TPM requirements so that none of a TPM's security is impacted by bad Software design, at the risk of a potentially poor user experience. As such, Software and drivers should follow these recommendations to ensure a robust implementation.

- 1) Software should continue to use the memory mapped `xxD4_xxxxh` address range to access the SPI-TPM.

NOTE: This is the same address range as for the LPC-TPM. All existing LPC-TPM code will continue to work as is with the SPI-TPM. Software which uses legacy LPC cycles to a TPM will not work with an SPI TPM.

- 2) The SPI TPM has added an extended data FIFO for larger data transfers. Software may or may not use this new register.

Software is recommended to continue using the existing protocols for accessing a TPM as described in Sections 6.5.2.4 and 6.5.2.5.

End of informative comment

10.1.3 Software Command Interface to SPI-TPM

Start of informative comment

Many platforms support FLASH components on SPI by allowing Software to program a command-based interface. Software sets up specific commands such as block write, erase, etc. through some platform dependent set of registers.

Note: some platforms may choose to not have a TPM as part of the platform or continue to use the LPC-TPM. Those platforms may use the SPI-TPM CS# for additional FLASH space or some other usage. In that case the command-based accesses may be allowed. The mechanism to determine whether the CS# is attached to a TPM must not be Software dependent but should be some HW mechanism fixed by the OEM at manufacturing.

End of informative comment

10.2 NV Index Provisioning for Platform Firmware Supported Features

Start of informative comment

The TCG PC Client Platform Firmware Profile version 1.06 specifies NV Extend operations to support capturing measurement of instance specific (such as a Certificate serial number or SMBIOS information containing serial numbers) information required to verify measurements extended to TPM PCRs. This section specifies the required attributes for these indices if they are pre-populated by a TPM Vendor or a PC OEM.

The TCG PC Client Platform Firmware Profile version 1.06 also defines an optional set of structures and a method using NV Indexes for event log integrity protection. There are two NV Indexes defined for this purpose: EVENT_LOG_INTEGRITY.EXIT_PM_AUTH and EVENT_LOG_INTEGRITY.READY_TO_BOOT. If the platform firmware implements this feature, both indexes need to be created in a secure manufacturing environment with the attributes specified below.

Note: The NV_Extend indexes should be provisioned in platform manufacturing. If the platform firmware is updated post manufacturing to support additional Hash algorithms or the Platform Firmware Setup menu allows alteration of the allocated PCR banks, the NV_Extend indexes might need to be reprovisioned by the platform manufacturer to reflect the revised Hash algorithms.

End of informative comment

The NV Index handles defined in Table 69 — PFP Defined NV Extend Indexes are for provisioning Platform Firmware Supported Features.

Table 69 — PFP Defined NV Extend Indexes

NV Index	Name	Purpose
0x01C40200	NV_EXTEND_INDEX_FOR_INSTANCE	NV Extend Record for instance data
0x01C40201	NV_EXTEND_INDEX_FOR_DYNAMIC	NV Extend Record for dynamic data
0x01C40202	EVENT_LOG_INTEGRITY.EXIT_PM_AUTH	Event Log Integrity for ExitPmAuth
0x01C40203	EVENT_LOG_INTEGRITY.READY_TO_BOOT	Event Log Integrity for ReadyToBoot

1. The NV_EXTEND Indexes (0x01C40200 and 0x01C40201) and the NV Indexes for Event Log Integrity SHALL have the following attributes SET: TPMA_NV_PPWRITE, TPMA_NV_PPREAD, TPMA_NV_OWNERREAD, TPMA_NV_AUTHREAD, TPMA_NV_POLICYREAD, TPMA_NV_NO_DA, and TPMA_NV_PLATFORMCREATE.
2. The NV_EXTEND Indexes ((0x01C40200 and 0x01C40201) SHALL have the following additional attributes SET: TPMA_NV_ORDERLY and TPMA_NV_CLEAR_STCLEAR and SHALL have TPM_NT_EXTEND
3. The NV Indexes for Event Log Integrity (0x01C40202 and 0x01C40203) SHALL be an Ordinary Index.
4. The NV_EXTEND Indexes SHALL be created with the Index nameAlg corresponding to the strongest algorithm (bit strength) among the allocated PCR banks.
5. The NV Indexes for Event Log Integrity SHALL be created with a Public DataSize = a 2-Byte Version + 2(Size of NV Index nameAlg). Note: This size is intended to accommodate the EVENT_LOG_INTEGRITY_DATA structure defined in the TCG PC Client Platform Firmware Profile version 1.06 and later.

11 References

1. The Trusted Computing Group: <https://www.trustedcomputinggroup.org>
2. Low Pin Count (LPC) Interface Specification: <https://www.intel.com/design/chipsets/industry/lpc.htm>
3. TCG PC Specific Platform Firmware Profile:
https://www.trustedcomputinggroup.org/developers/pc_client/specifications
4. System Management Bus (SMBus) Specification Version 2.0: <https://www.smbus.org>
5. PCI Express Electromechanical Specifications: <https://www.pcisig.com>
6. The Serial IRQ (SERIRQ) protocol definition: <https://www.smsc.com/ftpdocs/papers.html>
7. TPM Library Specification:
https://www.trustedcomputinggroup.org/developers/trusted_platform_module/specifications
8. I2C-Bus specification and user manual, Rev. 6, 2014-04-04, NXP
9. I2C CRC calculator reference: <https://reveng.sourceforge.net/>
10. TCG Glossary: <https://trustedcomputinggroup.org/resource/tcg-glossary/>
11. TCG ACPI Specification: <https://trustedcomputinggroup.org/resource/tcg-acpi-specification/>
12. PC Client Physical Presence Interface Specification: <https://trustedcomputinggroup.org/resource/tcg-physical-presence-interface-specification/>
13. Registry of Reserved TPM 2.0 Handles and Localities: <https://trustedcomputinggroup.org/resource/registry/>
14. TCG EK Credential Profile for TPM Family 2.0, latest version: <https://trustedcomputinggroup.org/resource/tcg-ek-credential-profile-for-tpm-family-2-0/>
15. PC Client TPM Interface Specification: <https://trustedcomputinggroup.org/resource/pc-client-work-group-pc-client-specific-tpm-interface-specification-tis/>